# A VLSI Shuffle Exchange Layout of Viterbi's Algorithm

by

Karl D. Mann

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science

Winnipeg, Manitoba, 1986

# A VLSI SHUFFLE EXCHANGE LAYOUT OF VITERBI'S ALGORITHM

BY

KARL D. MANN

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

ISBN 0-315-33873-3

# Abstract

Viterbi's Algorithm is useful in communication problems for combatting inter-symbol interference, and decoding convolutional codes. An implementation of Viterbi's Algorithm in CMOS silicon VLSI technology using the highly parallel Shuffle Exchange (SE) architecture is presented.

Two prototype decoder chips were designed and constructed to decode one source symbol per clock cycle. The first prototype has four processors, and operates at speeds up to 2.0 MHz. The second prototype is a two processor expandable chip set, designed to operate between 5 and 10 MHz. This prototype has not been tested, due to a wiring error.

The study verified that the SE layout is practical for solving Viterbi's Algorithm where decoding speed is important, and small problem sizes are involved. A method is developed to extend the practical problem size by partitioning the SE graph across chip boundaries, and wiring the remaining level of the SE graph on the printed circuit board level rather than at the single chip level.

# Acknowledgements

The author wishes to thank Dr. E. Shwedyk for his supervision and helpful suggestions and financial assistance during this thesis. The author also wishes to thank Dr. P. G. Gulak, whose work has been drawn upon innumerable times, and who served to clarify many ideas.

The author is indebted to colleagues in the University of Manitoba VLSI Design Group: C. Schneider and R. Schneider who answered interminable questions about design procedures, and P. Hortensius who kept the cell library in order and reviewed ideas as well as the thesis. Also, thanks are due to Dr. H. Card and Dr. R. McLeod for their encouragement in presenting portions of this work.

The author acknowledges the Natural Sciences and Engineering Research Council for financial assistance, and the Canadian Microelectronics Corporation for their loan of design and test equipment employed in this thesis. He is also grateful to Northern Telecom, Ottawa, for fabrication of the chips designed in this work.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1
# Introduction

For a finite bandwidth communications channel, intersymbol interference (ISI) becomes detrimental when the system is employed nearer its theoretical capacity. ISI manifests itself when channel output is affected by the current input symbol and the previous $v$ input symbols ($v$ is called memory length or constraint length). The effect is that the symbols are smeared together and the output can no longer be interpreted without the contextual information of the previous $v$ symbols. This is illustrated in Figure 1.

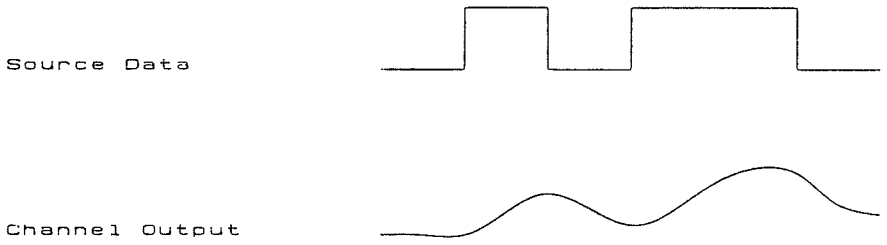


**Figure 1:** *Channel with ISI.*

Convolutional coding is often used with channels that have a high amount of noise, or are prone to burst errors. A convolutional code adds controlled redundancy to a symbol sequence. For example, an (n,k,m) code uses the current k symbols and previous k ×m symbols to encode a sequence of k source symbols into a sequence of n channel symbols. Because the channel symbols are each functions of past inputs, the decoder in the receiver must use the contextual information of the previous k ×m symbols to decode the current output symbols.

Viterbi's algorithm, which is described in the next chapter, is useful for solving both ISI and convolutional decoding problems.

Although the Viterbi Algorithm (VA) provides the basis for a good theoretical receiver, its complexity grows exponentially as the constraint length increases. The objective of this study is to show that by using a highly parallel architecture the VA can produce real time solutions to reasonably complex problems. Of the many parallel solutions described by Gulak [1], the Shuffle Exchange layout was chosen as having the greatest speed potential, while still being reasonably compact.

This work describes a design methodology for implementing the VA in a Shuffle Exchange (SE) layout, which requires $2^{\nu}$ processors to solve binary ISI problems, or $2^{m}$ processors to decode (n,1,m) convolutional codes. The prototype designs show that the resulting layout is practical with present technology, and meets the objectives of high speed decoding without using unreasonably large area. A strategy is described to expand the design to solve larger problem sizes. The design is structured and modular, and proves to be flexible in terms of building decoders with different design constraints. Also, the experience in laying out the design has allowed the constant factors in area-time complexities for VLIS referred to in [1], to be approximated for this architecture.

# Chapter 2
# Viterbi's Algorithm

This chapter discusses how the Viterbi Algorithm is used to solve intersymbol interference and convolutional decoding problems, and provides an illustrative example for each problem. Finally, it describes six ways in which the VA may be implemented in hardware and gives reasons why the Shuffle Exchange layout was chosen in the present work.

## 2.1. How the Algorithm Combats ISI

Viterbi's Algorithm combats ISI by taking into account the previously received symbols when determining the currently received symbol. Perhaps the easiest way to explain the algorithm is by considering a specific example. Consider a communications channel that has the impulse response shown in Figure 2, as measured at the sampled output of a matched filter. As shown, the channel output is affected not only by the current input, but also by the previous two inputs. The quantizer in the receiver has a resolution of 0.1 units. In terms of representing the channel output, Figure 3 shows how the channel can be modelled as a Mealy state machine [2], where the two previous input symbols represent a channel state, and the current input symbol represents a transition variable. (The noise and channel together are referred to as a Markov process [3], but the Mealy concept is useful later.)

For a binary channel, each state has two possible transitions. For example, if the two previously transmitted bits were (0,1), the channel would be in state 1. Receiving a 0 would cause a transition into state 2, while receiving a 1 would cause a transition into state 3, as the two most recent bits would be (1,0) and (1,1) respectively at the next bit interval. Thus there are eight expected channel output values, one for each possible three bit input sequence, as shown in Table 1.

If one represents the available states at an interval in time by a column of dots and draws in each valid state transition between that column and the next, one obtains one step of a trellis diagram for the channel as given in Figure 4. Each

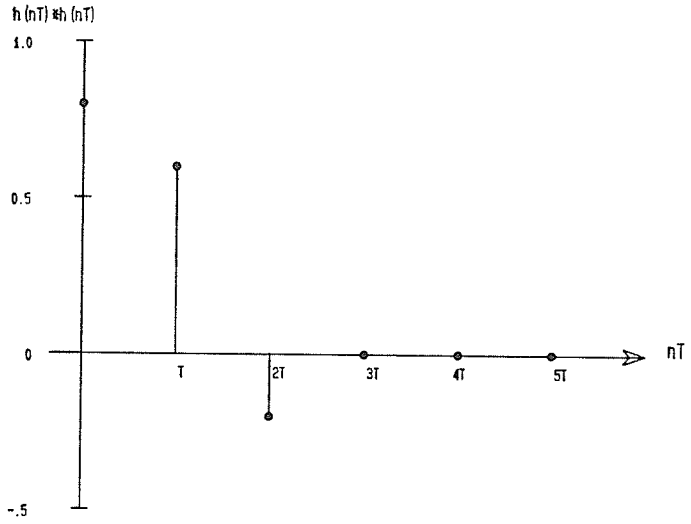


**Figure 2:** *Sampled Channel Impulse Response h(nT)*h(nT).*

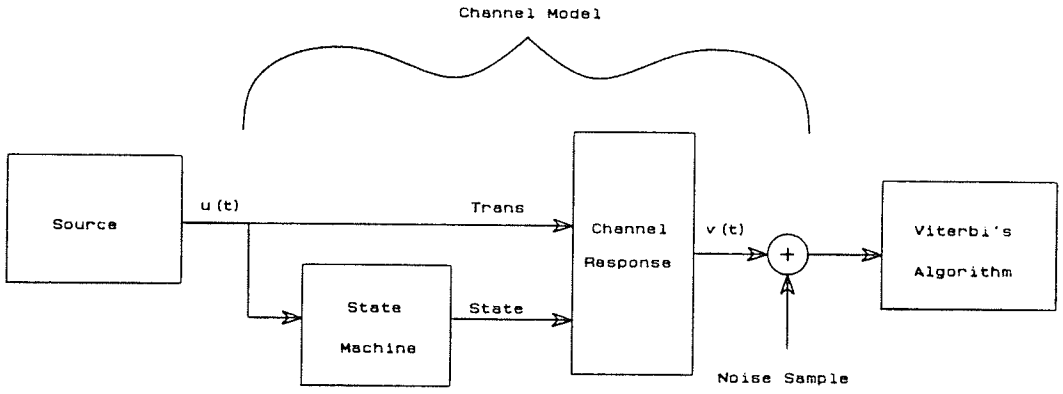


**Figure 3:** *State Model of Channel.*

transition has associated with it an expected value determined by the channel impulse response; designated by the v(nT) column in Table 1.

Because noise is introduced into the data sequence, the received values may differ somewhat from the ideal state transition values. This means that a received bit cannot be identified simply by picking a transition with a matching v(nT). Therefore a measure of closeness, or a metric, is used to indicate how well each possible

| u(nT-2T) | u(nT-T) | u(nT) | v(nT) | State | Trans |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0.0 | 0 | 0 |
| 1 | 0 | 0 | -0.2 | 2 | 0 |
| 0 | 1 | 0 | 0.6 | 1 | 2 |
| 1 | 1 | 0 | 0.4 | 3 | 2 |
| 0 | 0 | 1 | 0.8 | 0 | 1 |
| 1 | 0 | 1 | 0.6 | 2 | 1 |
| 0 | 1 | 1 | 1.4 | 1 | 3 |
| 1 | 1 | 1 | 1.2 | 3 | 3 |

**Table 1:** *Channel State Transitions.*

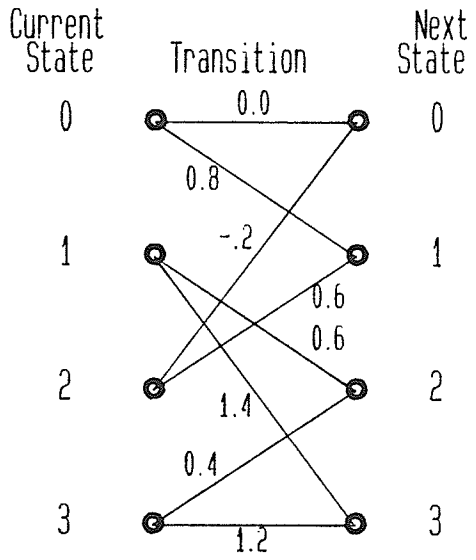


**Figure 4:** *Binary Trellis (M=2), v=2.*

transition value and a received value match. This "metric" need not fulfill all the metric axioms, but generally, a metric is 0 for a perfect match, and increases monotonically with error. In this example, an absolute value metric, $M_t = |rec - expected|$ is used.

Viterbi's Algorithm is a graph search procedure applicable to trellises, and determines an overall path through the trellis states least likely to be in error, i.e. the maximum likelihood path. This corresponds to finding a path whose transition metrics give the lowest sum, indicating that the accumulated error is a minimum.

Because an exhaustive search for such a path would take an impractically long time, the VA breaks the procedure into steps and maintains only $M^{\nu}$ paths at each step, where M is the number of channel symbols (M = 2 for binary) and $\nu$ is the memory length. For a particular time, each trellis state is assigned a state metric consisting of the sum of the transition metrics accumulated in reaching that state via the lowest cost path since time t=0. A record of the path is also associated with each node. At the next time interval, the algorithm compares the sums of the previous state and transition metrics for the M input paths on the trellis for each node. Of the M paths, the ones with the higher sums are obviously not part of the lowest cost path, so the path with the lowest sum is chosen, and the sum becomes the state metric for this node, while the other paths are thrown away. The path record (called a survivor sequence) is updated with this newest transition. When the whole sequence has been received, the node with the lowest state metric holds a record of the lowest cost path through the entire trellis.

Figure 5 shows an example transmission for the binary channel with memory length of two bits. The transmitted sequence is distorted with ISI by the channel response and then noise is added, resulting in the received sequence which appears above the corresponding trellis step in the figure. The transition metrics are labeled on the lines connecting nodes, while each state metric appears above the node.

Although for a short transmission it is possible to maintain a full record of state transitions at each node, it is readily seen that this would become impractical when decoding a continuous message. However, from observing the example in Figure 5, a solution becomes apparent. Each of the four states in the last trellis step agree on the first four path decisions. Since no paths have been retained which could have a lower cost at a later stage, these four steps will be part of the overall sequence which will be chosen when the message has been completely decoded. These decision values become the first four decoded bits. The four paths are said to merge and the history preceding the merge point may be extracted from the decoder. Therefore, the decoder is only required to keep a record of the decisions since the sequences last merged. Survivor sequences merge at random intervals, but simulation studies have shown that merging typically occurs within $5\nu$ trellis stages [4].

## 2.2. Decoding Convolutional Codes

Convolutional codes add redundancy to source data by convolving it with an appropriate coding transfer function. Each channel symbol is a function of several

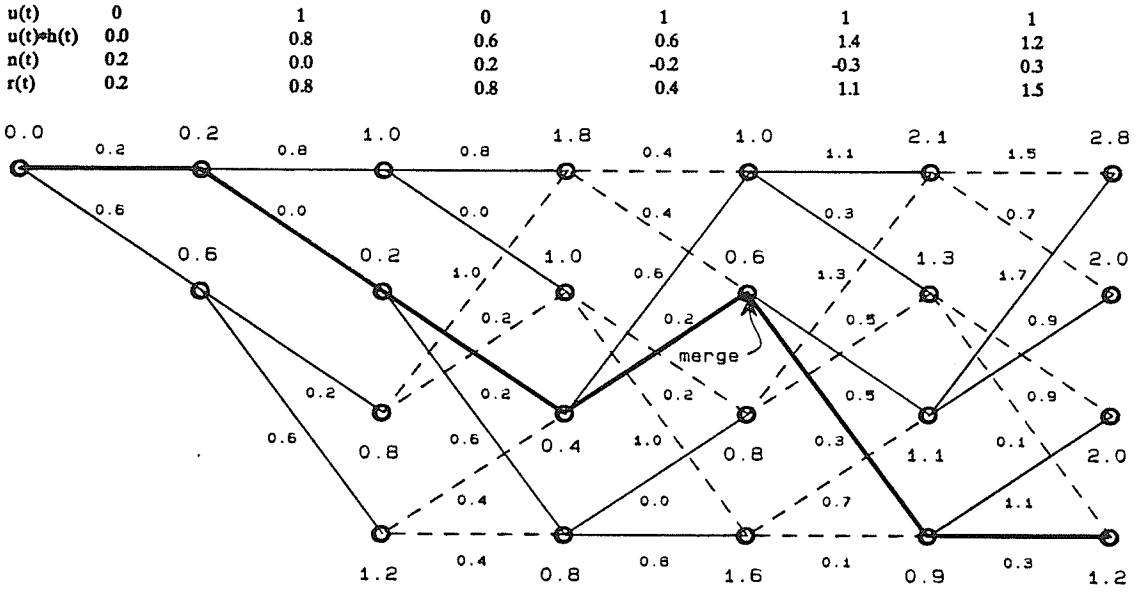


**Figure 5:** *Binary Trellis, v=2.*

past input symbols. An (n,k,m) encoder takes k input bits and shifts them into k separate shift registers of lengths up to m bits. It produces n outputs from these k inputs and their delayed predecessors. The channel then carries n/k times as many bits as the source produces. Lin and Costello [5] give a more thorough explanation of convolutional codes.

As an example, consider the (3,1,2) encoder in Figure 6 with the generator polynomial: $G(D) = [1+D, 1+D^2, 1+D+D^2]$, where D is the unit delay operator. Mathematically, the encoding may be performed by a matrix multiplication, v=uG, where u is an input sequence of length L, v is the encoded vector of length n(L+m), and G is an L by n(L+m) matrix derived from the generator polynomial:

$$G = \begin{bmatrix} 111 & 101 & 011 & & & \\ & 111 & 101 & 011 & & \\ & & 111 & 101 & 011 & \\ & & & \cdot & & \cdot \end{bmatrix}$$

If u = (110100) then L=6, and the encoded data would be $v$ = [111 010 110 100 101 011 000 000], where every bit in v is a function of three bits in u.

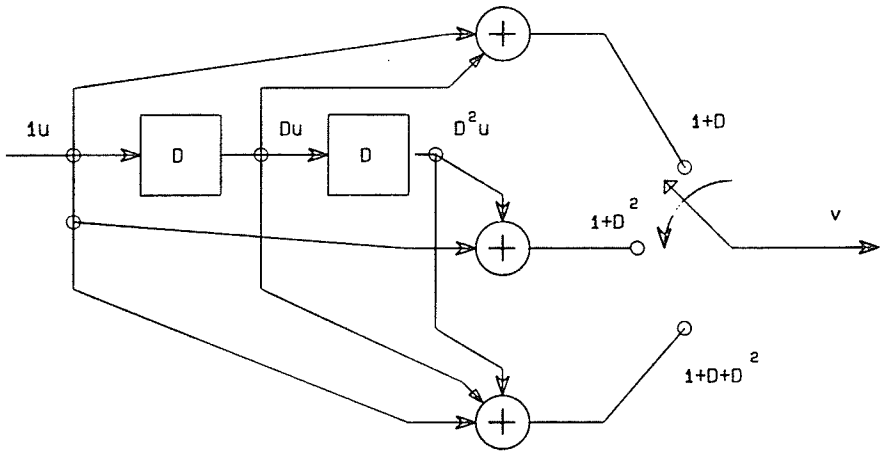


**Figure 6:** *(3,1,2) Convolutional Encoder.*

An alternate view of convolutional coding is to model the encoder as a state machine with $2^{mk}$ states, each having $2^k$ transitions to other states. As with the ISI example, let the state of the (3,1,2) encoder of Figure 6 consist of the two past input bits, and the transition variable is the current input. As a transition occurs, the n bit sequence marked along the state transition line in Figure 7 is sent into the channel.

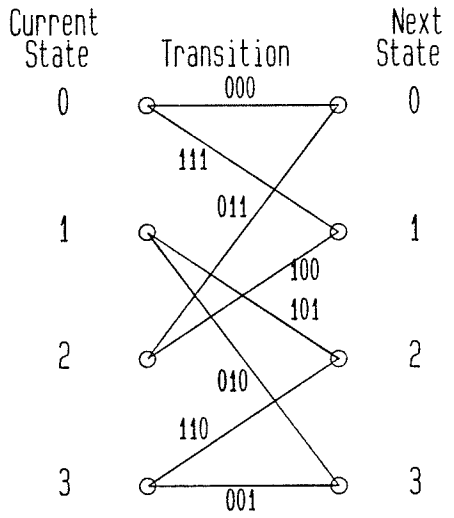


**Figure 7:** *(3,1,2) Code Trellis.*

The convolutional encoder and the ISI channel can both be modeled as a state machine. The similarity of the two problems allows the same decoding technique to be applied in both cases. If one draws the state transition diagram for each interval

in time, a trellis is produced. For each trellis step, the expected sequence for each transition is compared with the received sequence and a transition metric is assigned. Each state has a metric associated with it that is the sum of all transition metrics traversed since time t=0, using the lowest cost path into that state.

Continuing the above example, let the sequence u be encoded to give v. The transmitted values will be 10 units for a 1, and 0 units for a 0. Let v be corrupted by gaussian zero mean noise with a variance of 20 $units^2$. The received values are shown just above the trellis in Figure 8. In this case, let us use the Euclidean distance metric, $M_t = \sum_{i=1}^{n} (R_i - E_i)^2$, where each $R_i$ is a component of the received n bit vector and each $E_i$ is the corresponding expected component.

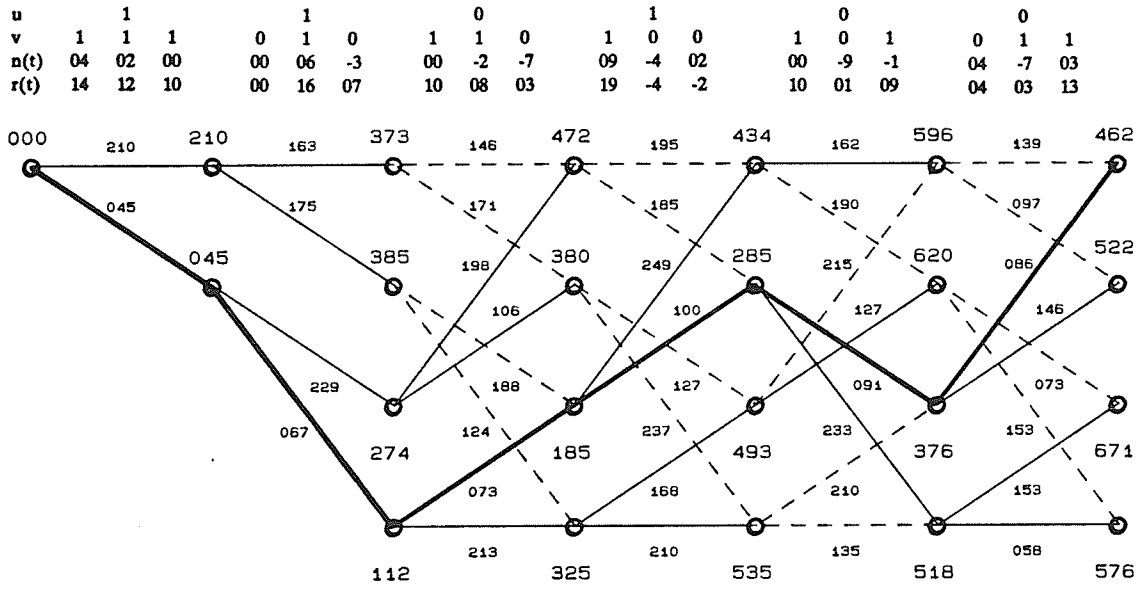


**Figure 8:** *(3,1,2) Code Trellis.*

As before, the state with the lowest metric represents the end point of the most likely path through the trellis. If we trace the path, we find that the sequence of decisions was indeed u=(110100).

This example is representative of all convolutional codes of type (n,k,m) where k=1. If k is permitted to be larger than 1, the trellis is no longer binary, and every state has $2^k$ possible transitions. The application of the VA to trellises of higher

values of k is dealt with by Gulak [1].

## 2.3. Parallel Implementations

The previous two sections have described how the Viterbi Algorithm works. However, nothing has been said about how it can be implemented or how quickly it can make decisions. One method of implementing the VA is with the Shuffle Exchange layout, but in order to understand how the SE layout fits into the broader scope of available layouts, an overview of several implementations is now presented. This description is not meant to completely explain the functioning of these alternative layouts. A more thorough description can be found in [1]. The notation used in this section corresponds to that developed for the ISI problem, unless indicated otherwise.

### 2.3.1. Single Processor

The algorithm can be programmed on a single processor machine, such as a general purpose microprocessor or a digital signal processing chip. Alternatively, specialized high speed VA hardware can be designed. The dominant part of a uniprocessor layout is the memory, which grows in proportion to $M^{\nu+1}$. As the constraint length increases, the uniprocessor is the smallest layout available. However, it is slower than a parallel implementation since it must perform $M^{\nu}$ Add-Compare-Select (ACS) operations in series for each received symbol (M=2 for binary).

### 2.3.2. Cascade

This layout depicted in Figure 9 uses $\nu$ identical ACS processors, with First In First Out (FIFO) queues of varying lengths between them to accommodate the routing of metrics and survivors. It processes $\nu$ symbols at a time, in $2^{\nu}$ time units yielding a net processing speed of $M^{\nu}/\nu$ time units per symbol. For small constraint lengths, the area is proportional to $\nu$, but as $\nu$ increases, the FIFO queues grow as $2^{\nu}$ and eventually dominate the area. The routing control used in this layout for the metrics is complicated. The cascade is easily partitionable into several VLSI chips if the need arises.

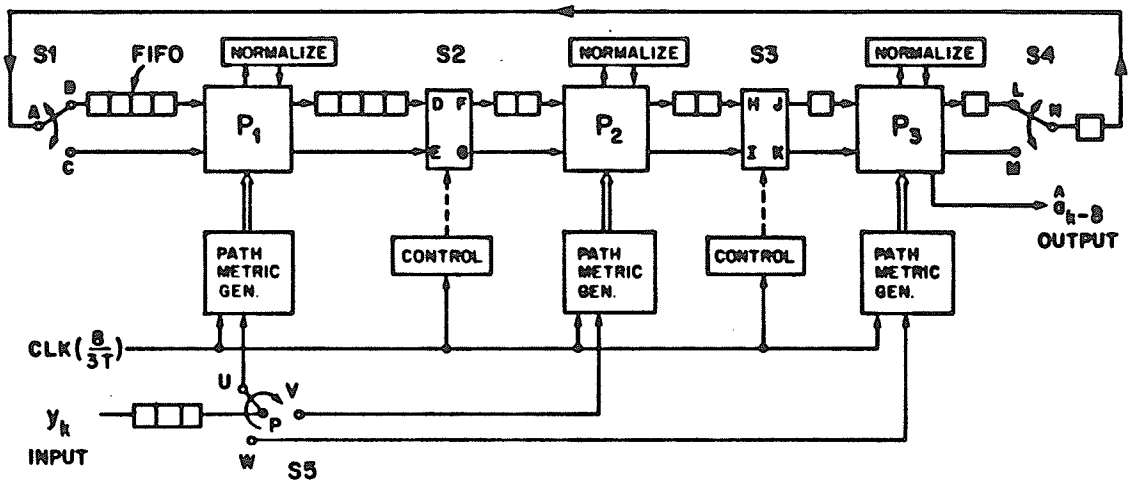


**Figure 9:** *Cascade Layout, $v=3$, $M=2$.*

### 2.3.3. 1-D Array

A set of $M^{v+1}$ identical ACS half processors can be joined in a line to form a systolic array as shown in Figure 10, with area proportional to $M^{v+1}$. The array is capable of solving the VA in $M^{v}$ time units per symbol. The only communication between units is the exchanging of metrics and histories with nearest neighbours. This layout has the advantage of easy partitioning if a multi chip system is desired, because only limited I/O is required at a junction between processors. However, most of the execution cycle is devoted to metric routing and only a small portion to ACS operations, yielding an inefficient design.

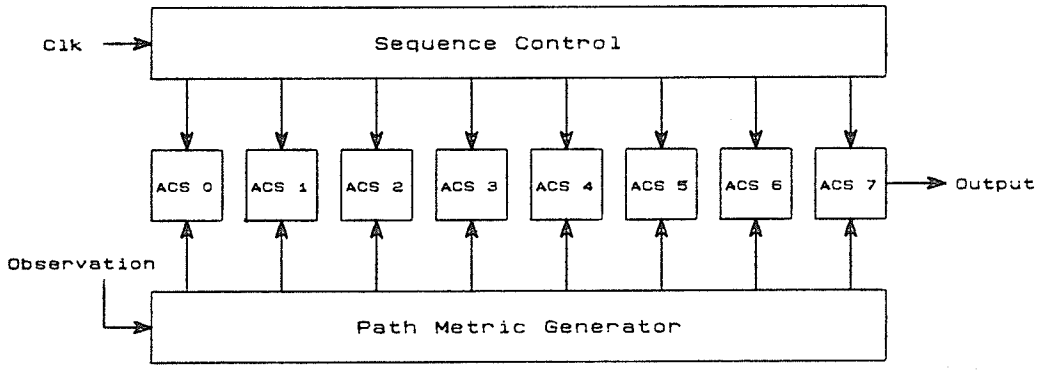


**Figure 10:** *1-D Array Layout, $v=3$, $M=2$.*

### 2.3.4. 2-D Array

The above idea can be modified to set up a two dimensional array of identical processors to solve the VA as shown in Figure 11. Communication occurs horizontally and vertically between nearest, and next to nearest neighbours. If it is possible to make use of the processors which are idle due to shorter information migration distances, the speed is lower bounded by $\frac{2^{\nu/2}}{\nu}$ time units per symbol. The control algorithm for this circuit would be quite complex, but the wiring for such an array of $2^{\nu}$ processors is compact.

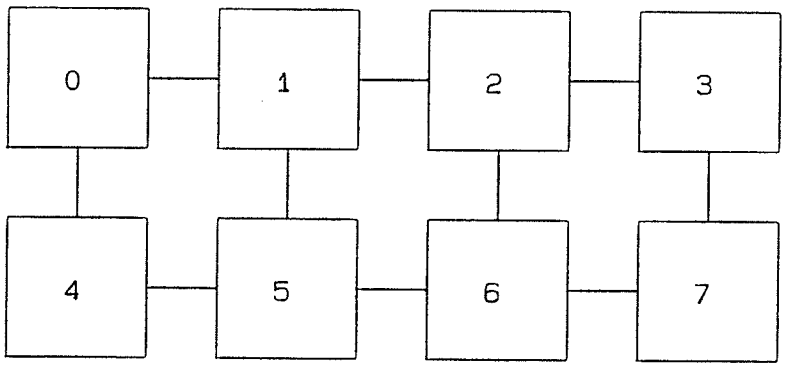


**Figure 11:** *2-D Array Layout, $\nu$=3.*

### 2.3.5. Shuffle Exchange

Complicated control systems can be avoided by connecting $M^{\nu}$ processors according to the SE graph in Figure 12. All communication is done in one step, allowing the algorithm to process data in one time unit per symbol. The connections are not nearest neighbour, so a larger wire area is used to wire past processors. This layout is partitionable to some extent, leading to a "generic" SE building block that is not restricted to a particular constraint length.

### 2.3.6. Cube-Connected Cycles

Rings of $\nu$ processors can be arranged on a $\nu$ dimensional hypercube as in Figure 13, thus using $\nu 2^{\nu}$ processors. Although it uses $\nu$ times more processors, the layout is more regular than the SE and thereby saves some wiring area. It can operate on $\nu$ separate data streams in one time unit. This layout is suitable for implementing functional blocks in separate chips and making from them large decoders of arbitrary

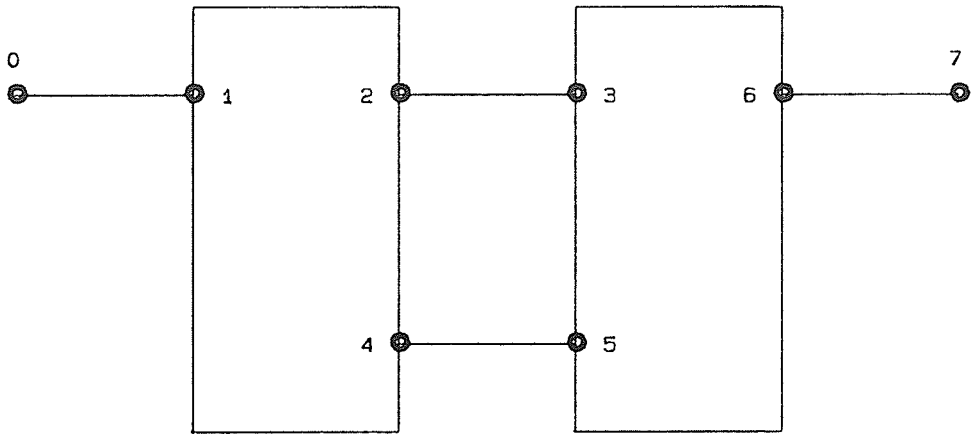


**Figure 12:** *Shuffle Exchange Layout, v=3, M=2.*

problem size on circuit boards. The Cube-Connected Cycles (CCC) layout can also be laid out in H-tree or Y-tree formation [1].

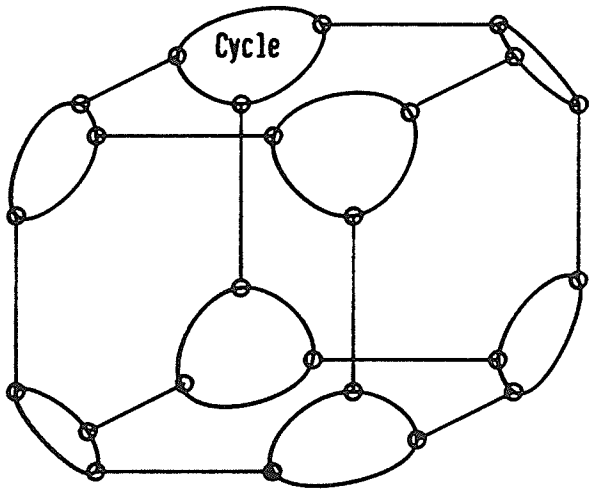


**Figure 13:** *Cube-Connected Cycles Layout, v=3, M=2.*

### 2.3.7. The Choice

Although each of the above layouts can be used to implement Viterbi's Algorithm, this thesis considers only one layout. Current implementation feasibility and high operating speed were considered important in the selection process.

The 2-D array was not chosen for study because to make it faster than the 1-D array, one must design a very complicated controller [1]. Some study should be done

to show whether the lower bound on symbol processing time could, in fact, be obtained, as the speed would make this an attractive solution.

The 1-D array was not chosen, because aside from a constant factor, it functions at the same speed as the uniprocessor solution, while using $2^{\nu+1}$ processors. In the systolic array, the area used by the processing elements dominates that used by the storage elements, while for the uniprocessor, the processing element could almost be neglected compared to the memory area.

The CCC uses $\nu$ times as many processors as the SE for a decoder of the same constraint length. Since the area for the SE circuit is nearly inadequate, a CCC is less practical than the SE is with the technology currently available.

The uniprocessor solution was not chosen for study because a simple solution already exists. A signal processing microprocessor, such as the Texas Instruments TMS32010, could be programmed to perform the ACS operations for each state in sequence. A greater throughput could be attained by using a specialized ACS processor, but the speed of operation would still be $M^{\nu}$ times lower than the SE. The uniprocessor is a good solution where high decoding speed can be traded off for higher memory lengths.

The choice of layouts was therefore reduced to the cascade and the SE layouts as providing the greatest area-time advantages. The cascade layout is a very reasonable method of solving the VA. It uses very little area, and is the second fastest implementation of those mentioned above. It bears closer study and was not used here simply because it is $\frac{M^{\nu}}{\nu}$ times slower than the SE. It also requires a more complicated control circuit than the SE, and is not as flexible when changing the memory length after a design has been completed. It is likely that if speed is not critical, this layout would be the most useful of all, because it would allow large problem sizes to be handled that currently cannot be attempted even with graph partitioning as described for the SE in the next chapter.

The SE was chosen mainly because it has the highest throughput of the six layouts examined. It is $M^{\nu}/\nu$ times faster than the cascade, and $\frac{2^{\nu/2}}{\nu}$ (where M=2) times faster than the lower bound on the 2-D array. The SE has no complex control logic and conforms naturally to solving the VA trellis. Also, if edge partitioning is used to construct a chip set, one generic chip can be used to solve the VA for many different constraint lengths. Its disadvantage is that the layout becomes impractically large except for problem sizes of small constraint length, $\nu < 9$.

# Chapter 3
# Shuffle Exchange Implementation

## 3.1. Representation of the Trellis as a SE Graph

One can see from the previous ISI and convolutional decoding examples, that solutions for each are obtained in a very similar manner. In these examples, the differences occur only in the manner in which a symbol or group of symbols is used to create a transition metric. To avoid confusion, the examples and terminology used from now on will correspond to the ISI problem, unless stated otherwise.

The trellis diagram of Figure 14 shows that at a particular time interval, several operations must be performed for each state of the trellis. For each of the two input branches, the state metric ($M_{s\,i,j}$) of the previous state must be added to the transition metric ($M_{t\,i,j}$) associated with that branch. The lowest sum is then kept, becoming the new state metric, and the state history associated with that branch is recorded.

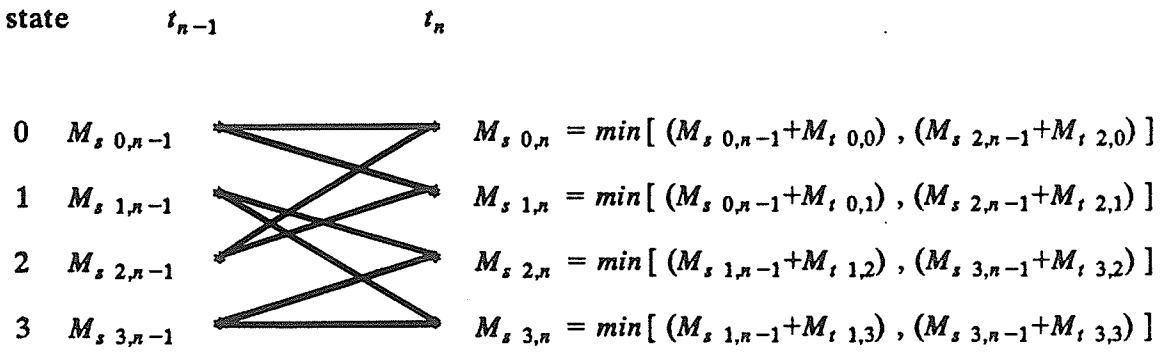


**Figure 14:** *Viterbi Trellis for $M=2$ $v=2$.*

Since the same set of operations is required at each state, why not associate a dedicated processor with each state, so that the work may proceed in parallel? The

input streams to each state processor carry the stored state metric and survivor sequence values of the previous time interval along the trellis branches. (A stream contains one state metric and one survivor sequence.) The receiving state processor adds the corresponding transition metrics and state metrics, and propagates the stream associated with the lowest sum. Once the decisions have all been made, they are stored and the state processors are used for the next time interval.

Therefore, each processor, as shown in Figure 15, must perform three basic operations:

1) Add the transition metrics to the sum of the previous metrics.

2) Compare the metric sums.

3) Select the lowest sum and record its corresponding state history.

These three operations constitute an Add-Compare-Select (ACS) operation. This method of parallel processing has the highest throughput possible for a single data sequence. Each received symbol is processed in only one symbol time interval.

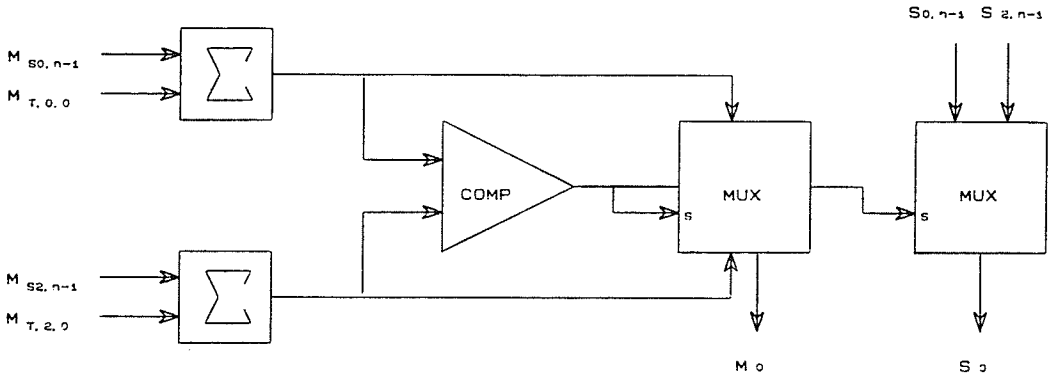


**Figure 15:** *ACS unit for state 0.*

A single stage of the trellis diagram used in the VA examples maps naturally onto the SE graph [6] described by Stone [7]. If each node in a Stone graph is represented by a unique $\nu$ bit binary number ( $\nu$ is the constraint length) $S_{\nu-1}, \ldots, S_0$ then two nodes N, N′, are linked by a shuffle edge if $N = S_{\nu-1}, \ldots, S_0$ and $N' = S_{\nu-2}, \ldots, S_0, S_{\nu-1}$, i.e., N′ is a cyclic shift of N. Two nodes N, N″, are connected by an exchange edge if N″ is a cyclic shift of N, followed by complementing the least significant bit (LSB). This arrangement of connectivity is isomorphic to one

stage of the VA trellis because of the nature of the problem; state changes are caused when new information is shifted into the channel, or the encoder memory.

Thompson uses another definition of a SE graph [8]. If each node in a Thompson graph is represented by a unique $v$ bit binary number $S_{v-1}, \ldots, S_0$ then two nodes N, N′ are linked by a shuffle edge if $N = S_{v-1}, \ldots, S_0$ and $N' = S_{v-2}, \ldots, S_0, S_{v-1}$, i.e., N′ is a cyclic shift of N. Two nodes are connected by an exchange edge if N′, N″ differ only in bit $S_0$. The collection of all cyclic shifts of a node N is called a necklace. A necklace that has $v$ nodes is said to be full, while a necklace with only one node is called a self loop. Figure 16b shows the resulting topology.

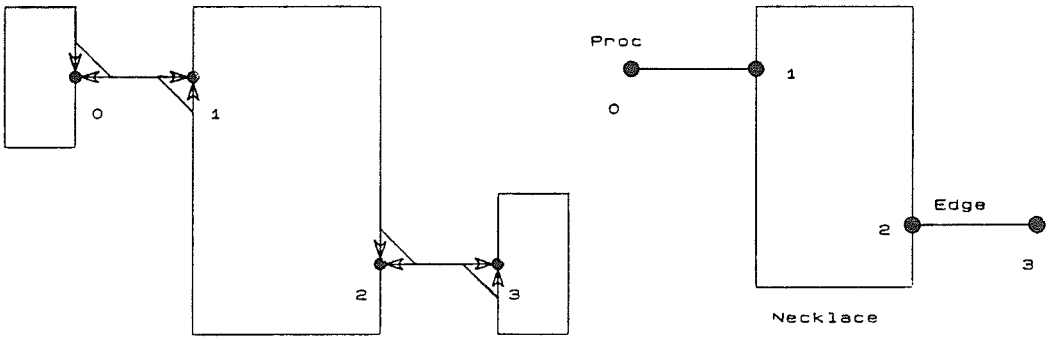


**Figure 16:** *a) Actual Connection* *b) Thompson SE Graph.*

It would be desirable to find a relationship between Stone's SE graph, and that of Thompson, because compact layouts have been found for Thompson's graphs [9]. It can be seen that Stone's exchange connection between N and N″ can be viewed as a shuffle from N to N′, followed by an exchange between N′ and N″. As a shuffle connection between N and N′ already exists, the information can travel along the same connection up to N′, whereupon the exchange connection bypasses N′ and its information travels on to N″. There is no extra delay in routing the message from N to N″ past N′, except for that associated with the incremental wire routing distance.

If the shuffle and exchange connections in the Stone graph are interpreted in this manner, it is isomorphic to the Thompson SE graph. This allows one to use the results in [9] to produce a compact layout. Figure 16a shows the actual connection of the processors according to the trellis, while Figure 16b shows the Thompson graph for the Shuffle Exchange.

## 3.2. Hardware Functions

As opposed to a network of general purpose processors, where protocols and queues are usually needed, the hardware for implementing Viterbi's Algorithm in a Shuffle Exchange is quite simple. Each state processor always receives two distinct input streams from two different processors. These streams are hardwired in the pattern of the SE graph and the only message control necessary is that all messages must be transmitted synchronously. This is done by storing each new decision output until the correct clock pulse issues the order for the outputs to take on the new values, and for the next set of decisions to begin.

The basic hardware for the VA processor in Figure 15 consists of several simple functional units. Each of the two input state metrics, one from the preceding processor on the necklace and the other from the partner on the edge, is fed along with its associated transition metric, into its respective adder. The two sums are compared, and a signal is generated to indicate the lower sum. This select signal controls two multiplexers, one selects the lowest state metric sum, and the other selects the corresponding survivor sequence. The outputs from these multiplexers are stored in the respective state metric sum and survivor sequence registers and then await the next decoding cycle. To avoid metric overflow, if the state metrics in all the processors have exceeded a defined threshold, normalization is performed before the streams are transmitted to the next processors at the next clock pulse. Also, a new bit is shifted into the LSB position of the survivor sequence, to keep track of this path decision. The oldest survivor sequence bit of each processor would be identical if a merge has occurred recently enough, so the oldest bit from one of the processors is latched to be used as the decoder output, while the most significant bit (MSB) from each of the other processors is simply dropped. Except for output storage, all functions may occur asynchronously, i.e., as data becomes available.

The transition metrics can either be calculated, or be looked up in a table contained in Read Only Memory (ROM) or Random Access Memory (RAM). If RAM is used, some method of initializing the contents must be provided.

## 3.3. Implementation Strategies

The Shuffle Exchange layout is attractive for VLSI implementation of Viterbi's Algorithm because there is a procedure for laying out the nodes which guarantees a compact design [10] that approaches asymptotic optimality for $v \geq 10$. For smaller graphs, a heuristic approach [9] produces more compact results. This approach is

based on ordering the necklaces from left to right so that the minimum value of the nodes in each necklace forms an increasing sequence. The necklaces are placed vertically, and the exchange edges are inserted between corresponding nodes in a way that minimizes the number of horizontal rows of connections.

If the entire design fits onto one silicon die, a minimum of Input/Output (I/O) is required. Aside from the power and control signals, only the attributes of the received signal upon which the transition metrics are based must be supplied and a single serial output is generated. As VLSI technology improves and the number of transistors on a chip increases, decoders for problems with larger $\nu$ will be accommodated on a single chip.

However, even as VLSI technology improves, there will always be applications which require a larger constraint length than the SE can accommodate on one chip. One is then forced to construct a receiver of a lower resolution or smaller constraint length to fit on a single chip, thereby increasing the error rate of the receiver. Alternatively, the problem could be partitioned to fit onto multiple chips and then combined on a circuit board to form a decoder of sufficient constraint length.

If a multichip configuration is chosen, one must immediately deal with time penalties inherent in crossing package boundaries on a circuit board. Because the ACS operation must occur simultaneously in all processors, if even one metric must cross a chip boundary, the ACS operation must synchronize with the slowest transfer. Likewise, the processing speed is limited by the slowest survivor sequence transfer. If multiplexing of I/O becomes necessary, the I/O speed can be severely limited. These physical constraint issues are not addressed by the VLSI grid model.

### 3.3.1. Functional Partitioning

There are three ways in which the total processing area may be divided into two or more chips. The metric calculation function and the manipulation of the survivor sequences are very loosely connected, so only the select line needs to communicate between the two functions. One may achieve functional partitioning by building two separate SE layouts, one for the ACS processors and a second for manipulating the survivor sequences. The total I/O would be the same as for the single chip version, with the metric calculation chip using received signal parameters, metric initialization and timing, while the survivor sequence chip would require only a clock input, each processor's select line, and would generate the serial output. As an example, two 40 pin packages have enough pins to implement a 32 processor, constraint length 5 system, and consequently share the area requirements. Multiplexing I/O and/or the use

of packages with more pins could extend this method provided the silicon area was available.

### 3.3.2. Necklace Partitioning

Obviously, one might consider an application where using two chips would still not provide enough silicon area to implement a receiver. In this case one must find a way of partitioning the SE graph in such a way that parts of the receiver can be built in separate chips, but that the inter chip I/O would still be feasible. It is important to partition the SE graph so that connections outside each package are minimized because the number of pins on a package is limited [11].

One method is to partition necklaces into chips, as depicted in Figure 17. This results in an average chip I/O of one input and output stream per processor contained on the chip, where a stream contains a state metric and a survivor sequence. For a constraint length $v$, this solution would yield a circuit board layout of at least $2^v / v$ chips, with $v$ processors in most of the chips [1]. Note that the number of processors per chip, and hence the constraint length $v$, is restricted both by the area available on one die and by the number of pins available for I/O in conjunction with how much multiplexing of the I/O is acceptable.

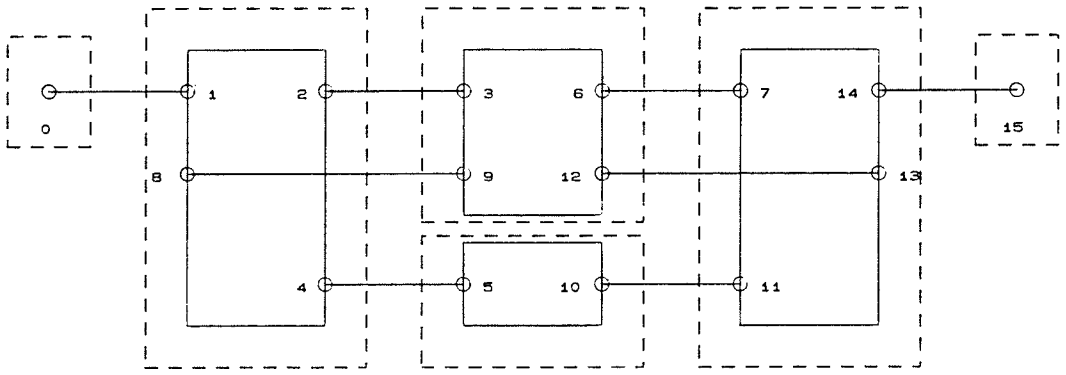


**Figure 17:** *Necklace Partitioning, $v$=5, $M$=2.*

A full chip set would include chips that are full necklaces, self loop chips, as well as any with partially full necklaces. To allow a chip to be used in any necklace position of the SE graph, it would be necessary as part of the initialization process to tell each processor whether it is an even or odd node, so that the correct bit may be added in the survivor sequence register each clock cycle.

Manufacturing two or more types of chips for a particular constraint length is expensive. To reduce the number of unique chips required for a decoder, one may replace the above specialized chips with one containing an entire necklace, but with a loop broken in such a way that the output of any processor may be fed back into the chip as in Figure 18. Although this necessitates using I/O pins for one extra input stream, one now has a general chip for constraint length $v$. For example, a self loop is made by looping the output of the first processor back into the chip instead of from the last.

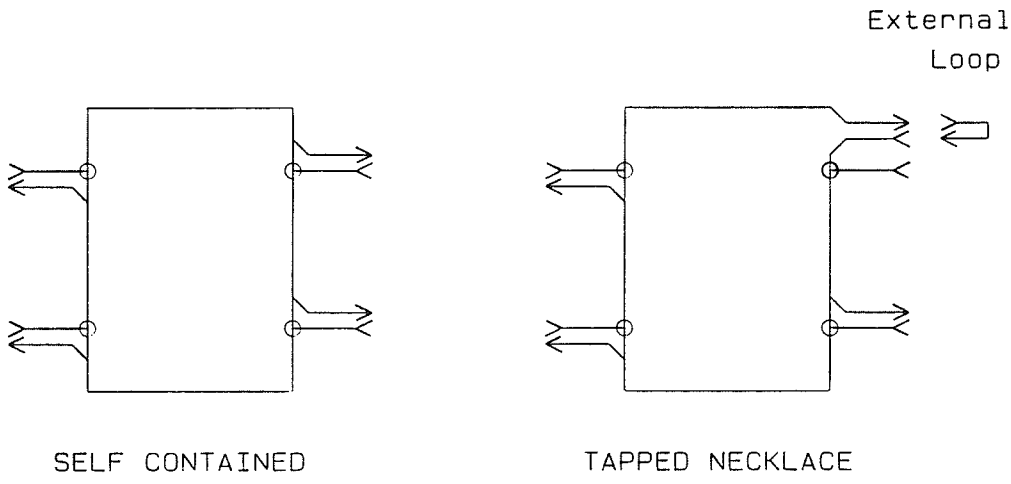


**Figure 18:** *Necklace Chips, $v=4$, $M=2$.*

The "generic" necklace chip also allows the construction of decoders with lower constraint lengths than $v$, because fewer processors need be used from each chip than are available. Alternatively, two or more chips may be coupled in series to build necklaces of more than $v$ processors. Thus, the cost of one extra input stream may be justified by a more flexible design approach.

### 3.3.3. Edge Partitioning

The final method chosen in this study, is to partition the SE graph along the edges as shown in Figure 19. Figure 20 shows how this method groups processors which have input streams in common onto a chip. A package containing a single processor would have two input streams and one output stream, but a package consisting of two processors on the SE graph connected by an edge requires two input streams, and two output streams, or an average of only one in, and one out per

processor. This eliminates some of the I/O involved in transferring streams between chips.

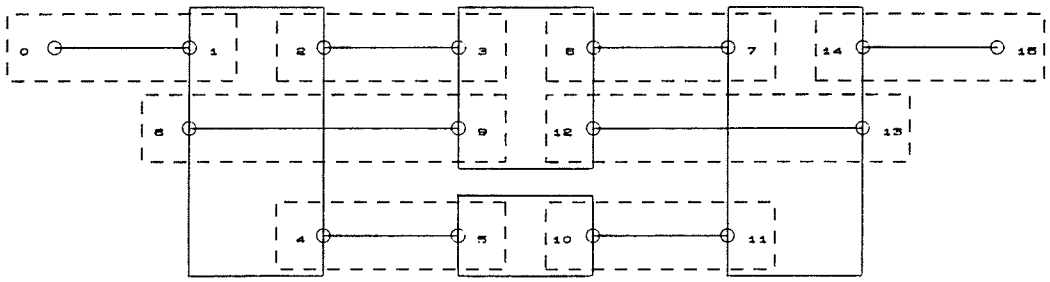


**Figure 19:** *Edge Partitioning, v=4.*

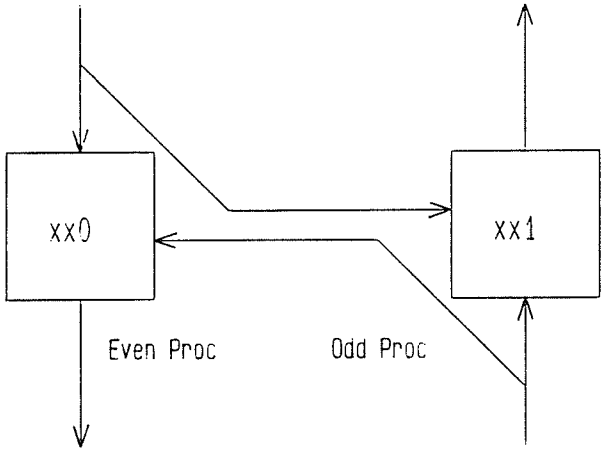


**Figure 20:** *Edge Chip.*

One advantage of this approach over necklace partitioning is that a chip manufactured in this manner is not constrained to solving a problem with a particular memory length. Now one simply constructs a shuffle exchange layout on a circuit board that looks like the SE graph for constraint length $v$ -1, using $2^{v-1}$ chips. This approach collapses pairs of SE nodes into single nodes of a SE of half the size as in Figure 21.

Unfortunately, this still leads to a high chip count, as only a 2 to 1 saving is achieved. Provided that chip area is available, one may iterate the collapsing procedure, and again collapse the edge nodes on the collapsed SE graph. This results in four processors per chip, again with an average of one input stream and one output

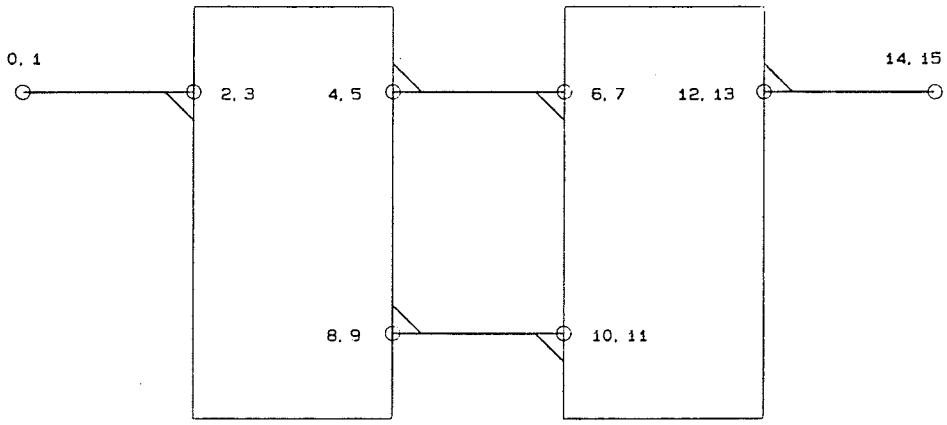


**Figure 21:** *Collapsed SE Graph for v=4.*

stream per processor in a package. The circuit board layout of the SE graph would use 1/4 of the nodes of the original SE graph, or $2^{v-2}$ chips (nodes). Like the two processor version, a four processor chip is not limited to solving problems of a particular constraint length (provided $v$ is not less than 2) and therefore is a good candidate for commercial manufacturing.

### 3.3.4. Proof of Collapsing Procedure

One may generalize the edge collapsing procedure for any number of iterations. It must be realized, however, that besides chip area, the number of pins available for stream I/O will limit how many processors may be put on a single chip. Only the first collapsing produces a net saving in I/O, while more collapsing merely reduces the chip count, while exhibiting a linear growth in I/O needs. The proof that each iteration results in a Stone shuffle exchange graph is now presented.

First define a binary shift right operator "$\rightarrow$" so that $C = A \rightarrow B$ takes A in binary form, and shifts it right by B places. During the shift, all binary places not originally within the bit width of A are assumed to be 0's. The result is truncated to the width of C. Similarly, $C = A \leftarrow B$ means that C becomes the left shift of A by B places.

To prove that the collapsed layout is also a SE graph, begin by denoting a processor, U, by a $v$ bit binary label, $U_i$, as in Figure 22. According to the Stone shuffle exchange graph [7], a processor $U_i$ receives input from processor $U_j$ if $j = i \rightarrow 1$, or $j = (i \rightarrow 1) + X$ where $X = 2^{v-1}$ (the MSB=1, all other bits 0). Let us define a set of processors, $S_x$, such that $U_i \in S_x$ for all i which satisfy $x = i \rightarrow K$. K is the number of collapsing iterations and x is a $v$ -K bit number, i.e. can be viewed as the upper $v$

-K bits of i, as in Figure 22. Therefore we have $L = 2^{\nu-K}$ sets

$$S_x: \quad S_0, S_1, \ldots, S_{L-1}$$

of $N = 2^K$ processors each. We would like to show that the new nodes $S_x$, will be interconnected according to a SE graph with L nodes.

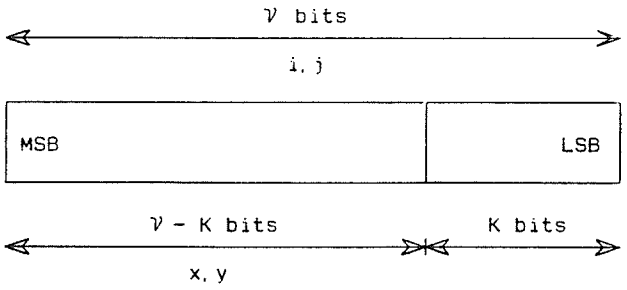


**Figure 22:** *Binary Label for $U_i$,$U_j$.*

If we examine the set $S_x$, where $0 \leq x < L$, we find that it contains processors $U_i$ where

$$i = \left\{(x \leftarrow K), (x \leftarrow K) + 1,\ldots,(x \leftarrow K) + (N-1)\right\}$$

Processor $U_j$ is an input to $U_i$ if $j = i \rightarrow 1$ or $j = (i \rightarrow 1) + X$, thus, $U_j$ is an input to set $S_x$ if

$$j \in \left\{ (x \leftarrow K) \rightarrow 1, [(x \leftarrow K) \rightarrow 1]+1, \ldots, [(x \leftarrow K) \rightarrow 1]+Y-1, \right.$$

$$\left. [(x \leftarrow K) \rightarrow 1]+X, [(x \leftarrow K) \rightarrow 1]+X+1, \ldots, [(x \leftarrow K) \rightarrow 1]+X+Y-1 \right\}$$

where $Y = L/2$ (the MSB of a K - 1 bit number = 1, all other bits 0). So the inputs to this set are $U_j$, where

$$j = x \leftarrow (K-1), x \leftarrow (K-1) + 1, \ldots, x \leftarrow (K-1) + (Y-1),$$

$$x \leftarrow (K-1) + (X), x \leftarrow (K-1) + (X+1),\ldots,[x \leftarrow (K-1)] + (X+Y-1)$$

We now wish to find from which sets, say $S_y$, the set $S_x$ receives its inputs. As defined above, a set $S_y$ consists of processors $U_j$ where $y = j \rightarrow K$. We can determine the set $S_y$ from the above list of j:

$$y \in \Big\{ [x \leftarrow (K-1)] \twoheadrightarrow K, [x \leftarrow (K-1)+1] \twoheadrightarrow K, \ldots, [x \leftarrow (K-1) + (Y-1)] \twoheadrightarrow K,$$

$$[x \leftarrow (K-1) + (X)] \twoheadrightarrow K, [x \leftarrow (K-1) + (X+1)] \twoheadrightarrow K, \ldots, [x \leftarrow (K-1) + (X+Y-1)] \twoheadrightarrow K \Big\}.$$

This reduces to $y = x \twoheadrightarrow 1$ and $y = x \twoheadrightarrow 1 + (X \twoheadrightarrow K)$. Note that

$$X \twoheadrightarrow K = 2^{\nu-1}/2^K = 2^{\nu-K}/2 = L/2 = Y$$

In other words, $y = x \twoheadrightarrow 1$, or $y = (x \twoheadrightarrow 1)+Y$. The Stone definition of SE defined processor inputs to be $j = i \twoheadrightarrow 1$, counting only the $\nu$ -1 LSBs. Thus, the sets $S_x$ can be laid out according to a SE graph of order $\nu$ -K.

An interesting point to note, is that in the original SE layout, the input to each processor splits in two, both entering a processor and traveling down the edge connection to its partner (because of the Stone SE definition). After a graph has been collapsed, this occurs in a slightly different way: an even numbered processor receives the first N/2 streams from its predecessor in the necklace, and sends the other N/2 streams to its odd numbered edge partner, as shown in Figure 23. The numbers in brackets refer to the node labels as they would be renamed for the collapsed SE graph.

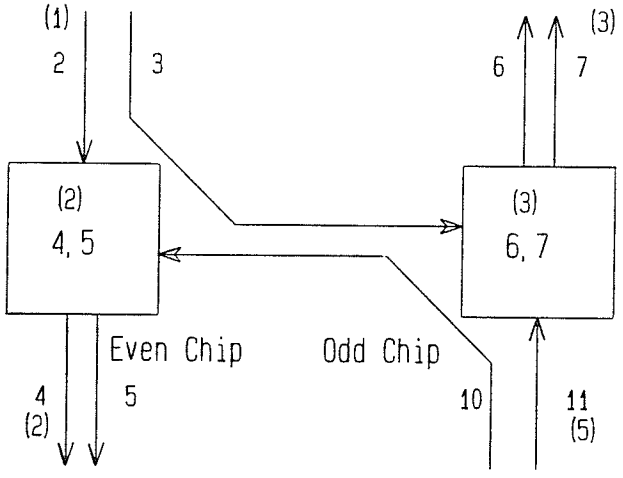


**Figure 23:** *Stream Splitting.*

### 3.3.5. The Choice

Both necklace and edge solutions produce a low I/O partitioning of the trellis. As technology improves and the ratio of chip area to processor area becomes larger, the factor governing how many processors can be placed on each chip will likely be the number of I/O connections each processor requires.

Edge partitioning was chosen for this study because in general, it is more flexible than necklace partitioning. The only advantage of using necklaces, is that as VLSI technology is refined, the number of processors per chip may be increased in quantities of one, while for edges, the quantities must always be powers of two. However, the edge method has the flexibility of manufacturing a single chip for all constraint lengths with a guaranteed circuit board layout, the SE graph.

Functional partitioning may be used in conjunction with either necklace partitioning or edge partitioning to reduce the area and I/O required by each method. More than half of each stream is devoted to transmitting survivor sequences between necklaces or edges. Removing this function to another set of chips and simply transmitting the select decision would alleviate some of the pin scarcity problem, as well as free up a significant portion of the chip area.

# Chapter 4
# Prototype Design and Testing

## 4.1. Design Tools

This section contains a brief overview of the procedures and tools used in arriving at a final chip design and implementation, and ultimately verifying its operation.

### 4.1.1. Programmatic Simulation

While trying to understand the operation of a Shuffle Exchange implementation of Viterbi's Algorithm, it was useful to write a computer simulation program. The resulting SE simulator was written in Pascal because it allowed modularity of functions and could manipulate data structures. This program served two purposes: firstly, it was used to "desk check" the author's perception of the algorithm, and secondly, it allowed simulation of the VA performance under various physical constraints.

The program consists of procedures which model the hardware constraints and capabilities. All operations are limited to the same resolution as the actual hardware being simulated, and a record is kept of performance degradation due to each approximation.

Input to the simulator is a simulated source with ISI and white noise. Parallelism is imitated through the careful use of data structures. The operational parameters of the decoder are adjustable, so it is possible to simulate different operating conditions, such as changing truncation widths, number of processors, survivor sequence lengths, etc. Experimentation with these parameters, coupled with the information about performance degradation is helpful in making design tradeoffs.

### 4.1.2. Electric

Electric is a VLSI CAD package [12] used to design integrated circuits at the University of Manitoba. Several of its features make Electric a convenient design tool. The design interface is efficient. The program is object oriented, which reduces the amount of knowledge the user needs about the VLSI design details, by allowing him to deal with transistors or cells, instead of the mask layers particular to the chosen fabrication technology.

As the design proceeds, Electric keeps track of the connectivity of the circuit and checks for design rule violations. These are often caused by placing two devices closer together than the fabrication process allows, creating a potential electrical problem. Electric gives immediate (i.e. on line) feedback to avoid potential problems.

Network lists suitable for the logic simulators can be produced from the design database, so at any stage of the design process the actual layout may be simulated. This gives feedback that the designer has indeed wired the circuit as intended, as well as verifying the logic.

From its design database, Electric prepares the Caltech Intermediate Form (CIF) listing, which describes the mask layers in detail. This is the information that is sent to the manufacturer to guide the fabrication process.

### 4.1.3. Logic Simulators

Three simulators are provided on the work stations used for this thesis. Mercury (HG) uses a simple transistor model to simulate the behaviour of a given network list. CSIM is simpler and treats each transistor as a switch. Both simulators use a network list produced by Electric to describe the circuit to be simulated.

It was found that HG is easily confused by large network lists and may thereby produce erroneous results. Thus, HG was used only for circuits that the switch level simulator would not understand, such as when two gates are in contention.

The simplicity of the CSIM model allows it to simulate large portions of a design and still yield a quick, and usually accurate, simulation. With both prototypes, the CSIM simulator was able to simulate the major components.

APLSIM, so named because of its APL based interface, is used when more accurate simulation is needed. This simulator uses a more detailed transistor model than HG, and takes gate and line capacitances into account to give an approximate timing simulation. This simulator was designed by R. Schneider at the University of

Manitoba, to be nearly as accurate as SPICE, but executes many times faster [13].

Input to the simulator is a network list derived from the CIF. Because it is based on the CIF description, the simulation is representative of the design as it will be fabricated, which may differ from Electric's net list.

APLSIM is valuable in testing parts of the circuit that have critical timing or are dynamic in nature, such as the metric RAMs. CSIM is not capable of handling dynamic circuits or of giving timing information.

The UNIX operating system used on the work stations allows pipelining of input and output files, so the simulation commands for any of the three simulators may come from a file, and the output data may be routed to an output file where further analysis can be performed.

#### 4.1.4. Design Rule Checker

All designs made at the University of Manitoba are submitted in CIF to the Canadian Microelectronics Corporation (CMC). There the designs are examined by a program called Dracula, which performs a detailed design rule check of the submission. Any errors are outlined and returned to the designer for correction and design resubmission.

#### 4.1.5. Test Equipment

If no design rule errors are found by Dracula and Electric, and the simulators verify that the logic and timing are correct, the design is then sent to CMC for fabrication. When completed, four samples are returned. These chips can then be tested at the University of Manitoba, to verify whether the design is sound and to determine its operating parameters.

To assist in this task, several pieces of test gear are used. An HP Data Generator and an HP Data Analyzer are used to apply various data sequences to the chip, and to compare the results with those expected. If problems exist, it is also possible to use micro probes under a microscope to access various test points on the silicon die to further analyze the problem.

## 4.2. Prototype Designs

Two prototype chips were designed and fabricated to test the validity of the SE layout ideas, and to measure the SE performance. In order to ensure testability, a

number of smaller test circuits were also fabricated, which could separately test the major components of an ACS processor.

### 4.2.1. Prototype CZZMB012

After experimenting with the Pascal Viterbi simulator, the next logical step was to construct a circuit to see how much area it occupied and how fast it could perform. During a preliminary layout to find the approximate size and shape of each cell, it was discovered that the maximum size receiver that could be built using 5$\mu$m CMOS on a 4800$\mu$m square die, was a 4 processor version (i.e. $\nu$=2) of limited arithmetic resolution. Simulation showed that 6 bits of survivor sequence length was usually to allow the sequences to merge before the serial output was generated. It also showed that a transition metric resolution of four bits was acceptable, provided that two guard bits were used to keep the sum from overflowing.

Since the hardware to implement this resolution occupied the entire die area, it was necessary to generate the transition metrics off chip and to use 8 four pin ports to bring them on chip. Although this meant 32 pins on a 40 pin package would be used up, it allowed the flexibility of using an external look-up ROM and so changes of the metric during experimentation were easy.

Although ideally the processors would be laid out according to Thompson's SE graph for four nodes, the limited space on the die accommodated only one layout; that of a pinwheel with all the interprocessor wiring crossing in the center of the chip. Gulak calls the SE design a high wire area layout [1] and the reason for this can readily be seen from the picture of the chip in Figure 24.

The operation of the chip is as follows. Each processor represents one state of the trellis diagram in Figure 25. As shown, there are two transitions into each state. Thus, each processor receives the two corresponding state metrics from the central crosspoint array, which carries the stored sums of each of the state's previous decisions. Meanwhile, the off chip hardware uses observations of the received symbol to produce a table address for both of the two transitions into that state. These four bit values arrive via buses from outside the chip. The state metric and the transition metric are added together in separate 4 by 6 bit ripple carry adders for each of the two input paths. The two sums are compared by a 6 bit comparator and a select signal is generated, indicating which sum is lower. While this is happening, the survivor sequences are also drawn from the crosspoint array. The 6 bit sequence selected by the above comparison is saved in a D-FlipFlop latch on the positive clock transition. The select line then gates the proper state metric into the normalizer.

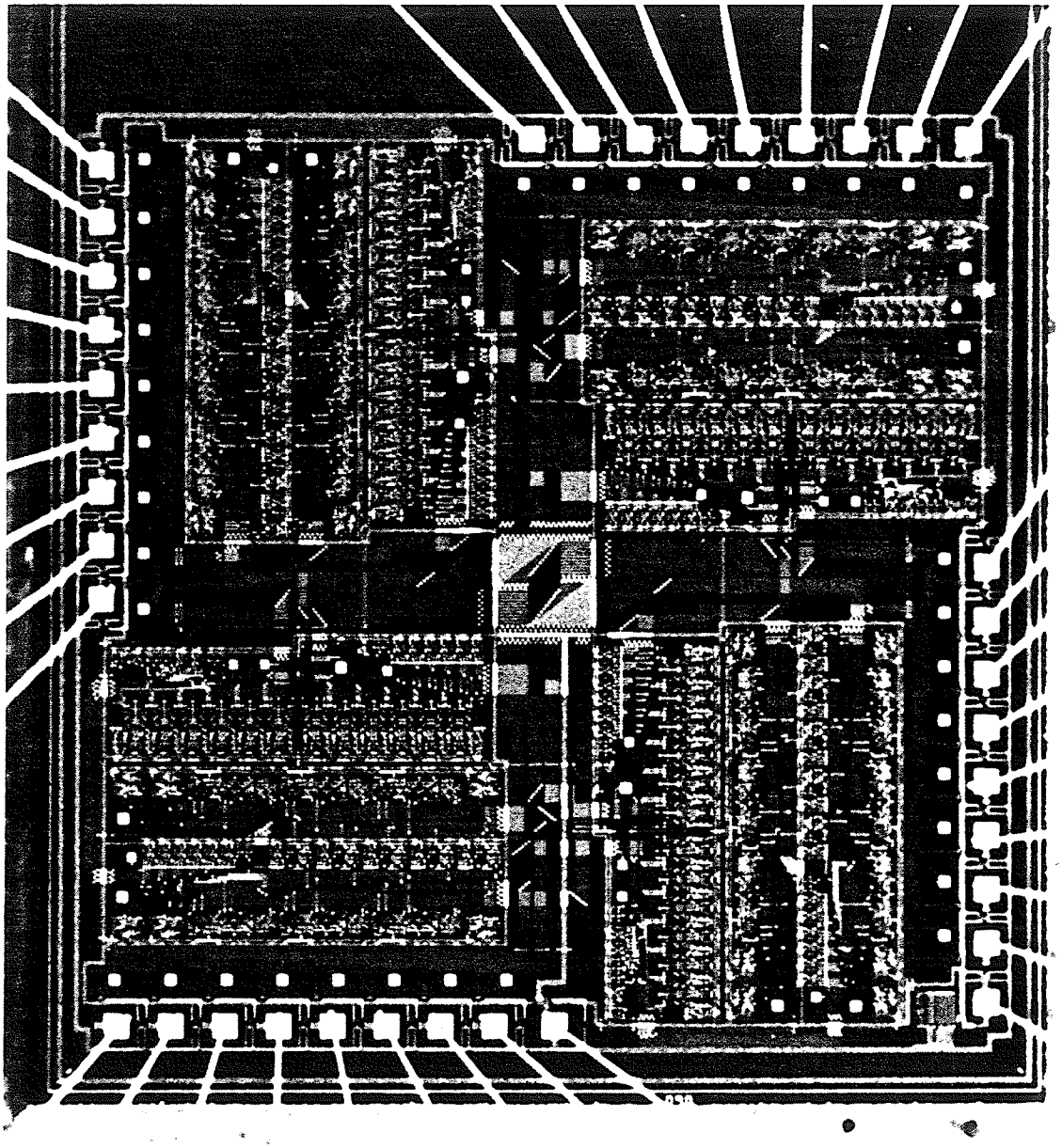

**Figure 24:** *Prototype Chip CZZMB012.*

Only if all four of the processors select a state metric with a value greater than 15, will each of the processors normalize its metric by subtracting 16 from it. This prevents wraparound overflow while involving only the upper two bits. On the negative transition, the 6 bit state metric is latched, thus replacing the information from the previous trellis step. The oldest bit of the survivor sequence is either presented as serial data or dropped, while all the other bits are shifted over by one position and the newest bit is appended to indicate the next trellis decision in the path

history. (The processor receiving the survivor sequence gets a pre-labeled sequence) This bit can be hardwired to a 0 for even processors, or 1 for odd. The procedure then continues with the next received symbol.

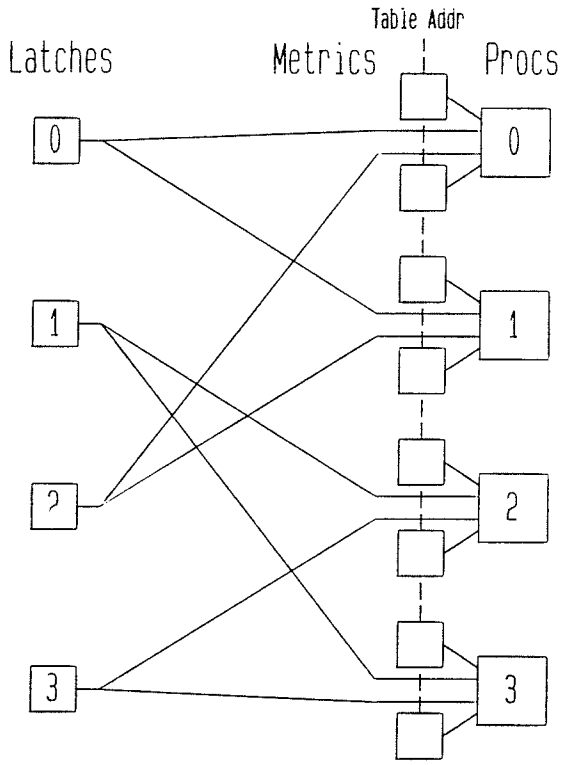


**Figure 25:** *Viterbi Trellis for CZZMB012.*

The CIF file was submitted to the CMC for fabrication in the 5μm Northern Telecom CMOS 1B process in May 1985, and samples were received in September 1985. Tests have shown that the circuit performs as designed. More specific information about the hardware can be found in the appendix.

### 4.2.2. Prototype IC3MB023

In order to obtain a prototype Viterbi chip that is flexible enough to use for various memory lengths, one must either have enough die area to make a receiver for the largest memory length of interest or use necklace or edge partitioning and then design the receiver on the circuit board level. Since the technology available at fabrication time did not allow the former, the latter option was elected. Edge

partitioning was chosen for reasons discussed under "Implementation Strategies".

The size and resolution of the second prototype chip was decided based on the number of pins available, namely 40. Two processors required two input and two output streams of data, plus timing, symbol values, serial output, power and control pins. By multiplexing two signals onto each metric or survivor sequence line, each stream could be composed of 10 bits of survivor sequence, and four bits of state metric, for a total of $(4+10)\times4/2=28$ pins. This left 12 pins for the other functions. Therefore, without using packages with more pins, or moving the survivor sequence processing to another chip, only two processors could be placed on a chip. This corresponds to one edge in the original SE graph. A plot of this prototype is shown in Figure 26.

Because the task of wiring between functional units for the original design was time intensive, the second prototype was based on bit-slice cells. These could be combined to any bit width simply by butting them together and making the proper connections. Then the major wiring job consisted of interprocessor wiring, and connecting processors to the multiplexers and demultiplexers.

Instead of using two guard digits in the ACS adders, the sums are clipped to avoid wraparound. In other words, any number over 15 would be limited to 15 after the operation. The carry bit from the addition is used to make the comparison operation five bits wide instead of four. It was found by simulation that because the ACS operation selects the lower sum, the one that overflows is usually ignored. Even if the lower sum also overflows, this only leads to a moderate increase in the number of decoding errors, as an overflowed sum is not likely to occur on the maximum likelihood path.

An additional enhancement became possible when access to 3$\mu$m CMOS became available through CMC. There was now room on the chip for two exchange edge processors and their transition metric look-up RAM. This RAM was configured as four 16 location by four bit static memories. Keeping the metrics in RAM allows one chip to serve in any of the edge positions, even though each chip needs different table contents. As a consequence, each chip requires RAM loading circuitry so that a whole circuit board full of Viterbi SE edge chips can be initialized.

Several things have been done to streamline the decisions made by the processors as shown in the timing diagram in Figure 27. A limited amount of pipelining is used, in that the transition decision leads the survivor sequence latching by half a clock cycle. Also, since the adders are ripple carry, it is important to have the lower bits of the previous state metric arrive first, so that the sum may begin forming,

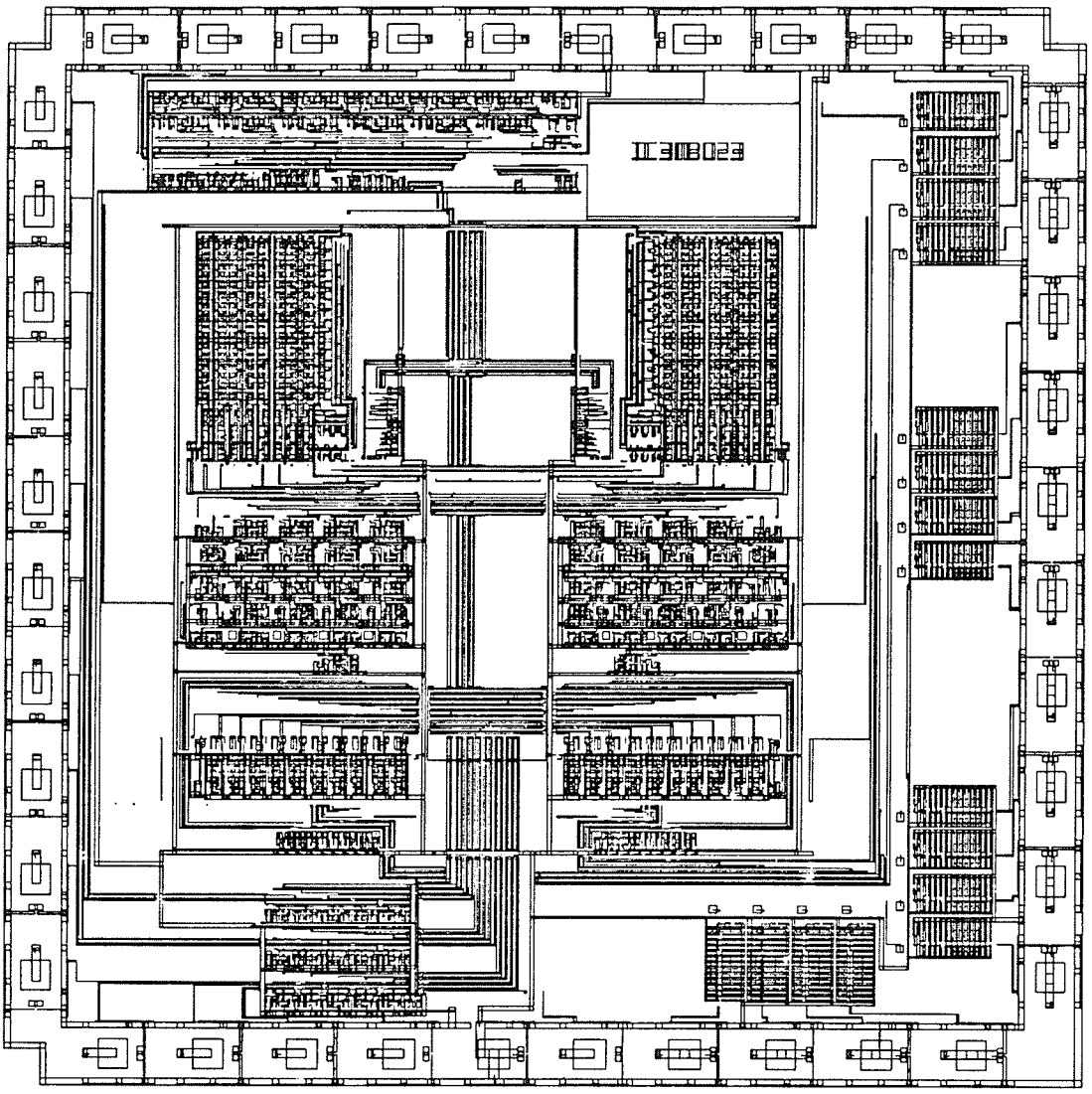

**Figure 26:** *Prototype Chip IC3MB023.*

while the upper bits are multiplexed during the second half clock cycle. Since normalization affects only the upper bits, it can take place after the sum is latched, rather than before, so the operation is performed while the lower bits are already being transmitted over the multiplexed bus. This also allows the normalization decision, made off chip, to complete after the sums are already latched, avoiding the bottleneck found in the previous prototype.

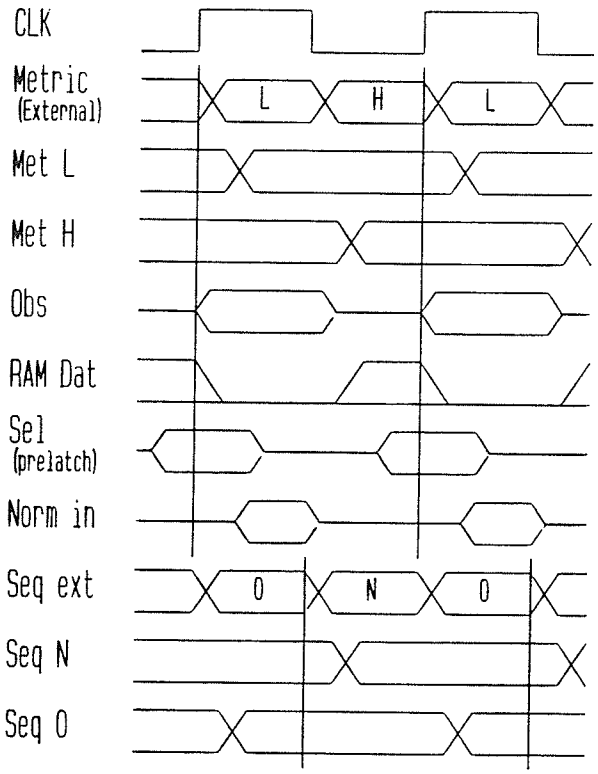


**Figure 27:** *IC3MB023 Timing Diagram.*

It was intended to obtain 16 samples of this chip, thus allowing a 32 processor implementation of a SE receiver to be built. The power of such a receiver is not trivial, and would demonstrate the potential of both the SE layout and of edge partitioning. Due to an error in connecting the clock to the ACS processor, the prototype as submitted in January 1986 will not function properly when fabricated. Thus, a second release of this prototype is to be submitted in May 1986, with this error corrected. The objective is that the prototype will operate in the 5 to 10 MHz range.

More detailed information on the operation and loading of the edge chip may be found in the appendix.

## 4.3. Simulations

There were two aims in simulating the designs. The first was to confirm that the logic indeed performed the desired function. The second was to confirm that

there were no careless wiring errors in or between functional blocks.

The simulation proceeded in a hierarchical manner. After each functional unit or macro cell had been laid out, a test was performed to verify its operation. This usually involved constructing a command file for the simulator to apply an exhaustive combination of inputs, or at least a unique sequence of inputs that would test all areas of the system's response. The results were manually compared with those expected.

Once larger functional units were constructed from the basic macro cells, simulation was used mainly for wiring verification. A unique pattern of inputs and outputs was used to distinguish whether bits in a bus had been reversed, that timing was in phase and other such details were not forgotten during the design process. If the individual macro cells worked under exhaustive testing and the larger units appeared to respond correctly, it was unlikely that a wiring error existed.

The logical conclusion of this exercise was to simulate the entire chip at the top of the design hierarchy. Unfortunately, this was never possible, as the amount of circuitry, coupled with the large amount of recirculation inherent in the SE graph, made the job too big for the simulators to handle. Limited success was obtained with the CZZMB012 prototype, in that if one could somehow force the internal voltage representations to initialize to reasonable values, parts of the ACS operation could be simulated. Enough results were obtained to ensure that the bonding pads were properly connected to the processors. With the IC3MB023 prototype the survivor sequence processing could be tested with CSIM, provided one examined and later manipulated the demultiplexed input bus, thereby testing half of the logic at a time. The metric handling operation yielded ambiguous results, possibly because of the clock connection error. The RAM loading and operation could only be tested by APLSIM because of its dynamic nature. For this test, the non RAM parts of the design had to be removed to reduce the size of the simulation.

One important advantage of testing by simulation, besides the obvious reason that it is cheaper to find errors in advance, is that internal nodes can be examined. Points which normally do not communicate with the outside world are available through simulation, and can be viewed or manipulated directly to aid in trouble shooting.

## 4.4. Chip Tests

The prototype designs were fabricated by Northern Telecom. Both the 5$\mu$m and 3$\mu$m channel length processes have single metal and single polysilicon layers. Four samples of each chip were returned for testing.

The hardware testing follows procedures similar to those used in simulation. The main difference is that with the hardware, one cannot usually see what occurs inside the circuit except where pins are connected, or through the use of probe pads, where they are provided.

### 4.4.1. Prototype CZZMB012

The first test performed with the sample was to measure the current consumption as a function of the supply voltage. The HP parameter analyzer provided a current limited voltage source for this test and showed that the power supply lines were not shorted internally.

Secondly, the test circuit in Figure 28 was built using four 2716 EPROMs for the 8 four bit look-up tables. The HP Data Generator was used to supply four bit data to the circuit, while the output was viewed using the Data Analyzer.

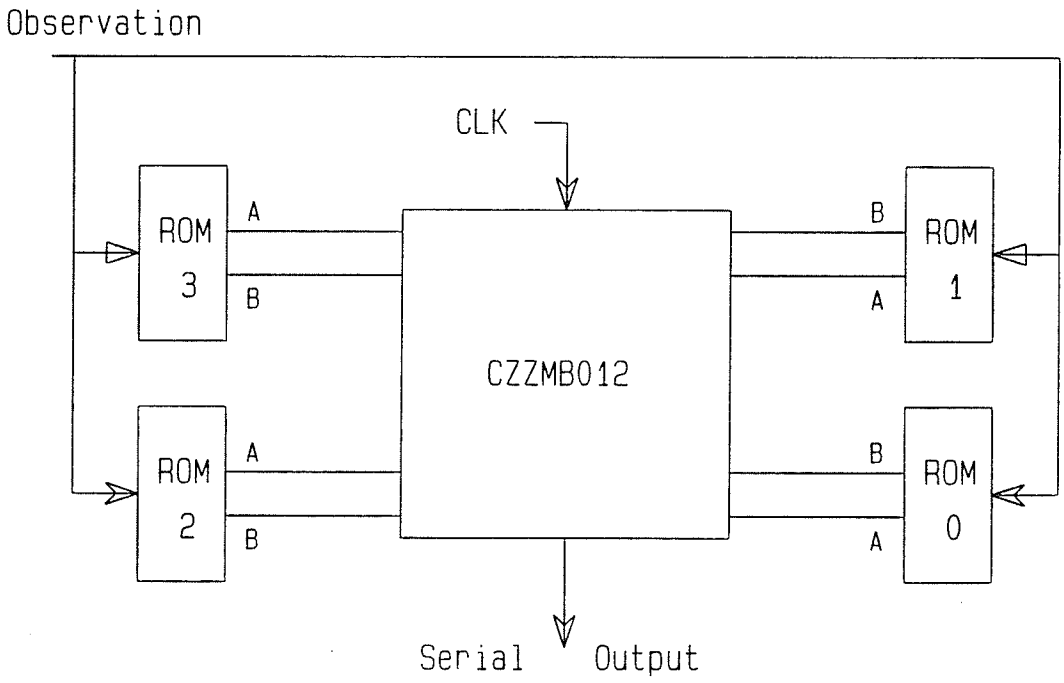


**Figure 28:** *CZZMB012 Test Jig.*

A simple test pattern was used to ascertain whether the chip might be making the correct logic decisions. Four separate periodic paths through the trellis were chosen: 0000..., 001001..., and 011011..., and 1111.... To produce these paths with a minimum amount of effort and control the amount of normalization that occurred, four short sequences of artificial branch metrics were loaded into the EPROMs of the test jig. The necessary input sequences were from one to three steps in length and were artificial in that they did not arise from a physical problem.

The 001 pattern will serve as an example. The trellis path is shown in Figure 29. If on the first stage, the transition from state 0 to 1 receives a 0 metric, while all other transitions receive a 2, and likewise, on the second and third stages, transition 2 to 3, and then 3 to 0 receive 1's, while all other transitions get 2's, one can see that the least cost path through the trellis will have the pattern shown. The initial values in the registers are of no consequence, as the path will converge to this pattern after several trellis steps. With this periodic sequence of length 3, the lowest metric sum will build up to 16 every 8 periods, at which point all the sums will be normalized. The amount of normalization can be varied by changing the value of the branch metric associated with the lowest cost path.

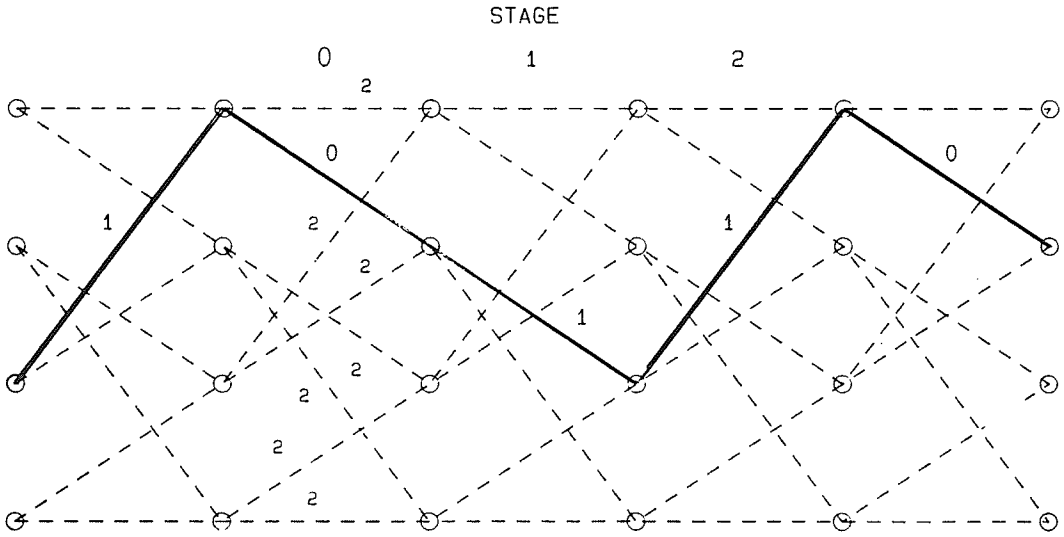


**Figure 29:** *001 Pattern Test.*

The four patterns were used to evaluate the prototype, and it was found to produce the proper pattern in each case, provided that the chip was not run at more than 2 MHz. With a sequence that did not build up a metric sum, the chip was

found to produce an accurate pattern up to 10 MHz. This indicates that the normalization circuitry is at fault for limiting the practical speed to 2 MHz, which is not surprising, as the decision to normalize is based on a wired OR connection between all the normalize request lines. This dependency was not discovered in advance because APLSIM was not yet available, and HG and CSIM do not give timing information.

Admittedly, this is not the most comprehensive test that a receiver might encounter. The data sequence used earlier with the Pascal simulator was applied, using the transition metrics shown in the appendix. It was found that the circuit accurately decoded the input sequence at an operating speed of 2 MHz.

At this point, the clock timing was varied, to see how short the respective cycle components could be made. Because no significant events take place between latching the survivor sequences, and latching the metric sums, the clock need only stay high for about 50 nanoseconds. The serial output, although delayed internally by about 100 nanoseconds, had a short rise time with an oscilloscope and the Data Analyzer as a load.

### 4.4.2. Prototype IC3MB023

The second prototype has not been manufactured at the time of writing, so the following is a test procedure description rather than a presentation of test results.

A more comprehensive test is possible with this prototype because all the survivor sequences and state metrics are available outside the chip. Also, the increased complexity of the multiplexers and RAMs make such a test desirable. In following this procedure, it is assumed that if some function is found to be in error, that the rest of the procedure will not be followed blindly, as an error in one area may affect the results while testing another. This procedure will work as described only if the prototype operates as designed.

To ensure that the RAM contents initialize properly, the following test can be run. Load the RAM with a unique pattern of bits. Then, operate the chip with the inputs to the state metric ports set to zero. Sequencing though the 16 possible channel values will cause the RAM nibble at that address with the lowest value to appear as the state metric sum output. To check the value of the other nibble at each location, one may increase the sum inputs corresponding to the lower valued nibble to cause the multiplexor to select the other value. If there is any doubt as to which nibble each processor chooses, the survivor sequence inputs may also be tagged by applying all 0's to input 0, and all 1's to input 1. The output survivor sequence bits

will then be equal to the select line value in each respective processor.

Having proven that the RAM contents read back as entered, one may test the ACS operation. Notice that if this operation does not work at least partially, the above test cannot work. Load the RAM with known values and apply metric inputs. Ensure that the metric outputs are equal to the lowest sum of either input 0 and RAM nibble 0, or input 1 and RAM nibble 1. Again, the survivor sequences can be used to tag the selected sum.

To test the survivor sequence handling, load the RAM with transition values that will pick a predetermined path through the trellis with minimum effort. Use sum inputs of all zeros and choose unique patterns for the sequence inputs. By stepping through the trellis, one can see that the correct sequence is chosen, and appears shifted by one bit. The serial data should be equivalent to the sequence bit that has just been dropped from the chosen register.

If the chip works entirely, a test jig can be set up to give it some real work. Two chips may be wired to make a four state decoder. This test can be used to evaluate the performance of interprocessor communication, as each bus crosses a chip boundary. For a chosen trellis, the transition metrics can be generated and loaded into the RAMs. The circuit should be fed realistic test data, whereupon the output should agree with that created by a simulation.

## 4.5. Area, I/O & Performance Results

The results of the simulations, design work, and prototype tests are the following:

- A single chip, four processor working prototype was produced.
- A two processor exchange edge receiver subsection was designed, and is ready for fabrication.
- A cell library of Viterbi components has been designed to facilitate present and future designs.
- The number of pins available for I/O is a serious restriction when partitioning a SE graph. A 40 pin device package allowed metric buses of four bits, and sequence buses of 10 bits, with multiplexing. For more processors on a chip, or for more resolution, it is necessary to use a package with more pins, or to multiplex more signals on a pin.

- When partitioning, as soon as a package boundary is crossed, the entire decoder must be paced to the slowest crossing. Conversely, if more chips are used on a circuit board, the receiver does not slow down appreciably.
- Though in general survivor sequences merge within $5\nu$ trellis steps, the software simulator showed that for a channel with the impulse response assumed, six steps, or $3\nu$ were usually adequate.
- The simulation also showed that one could avoid overflow problems in a four bit ACS processor by either providing two guard bits, or by limiting the sum when it reached the maximum value, and using the carry bit to make a 5 bit comparison.
- A single chip decoder for memory length 2, was able to decode a symbol every 500 nanoseconds. With the elimination of a known bottleneck, this could have been accomplished in 100 nanoseconds.
- The size of a four bit processor without metric generation hardware was found to require a grid space of about 480 $\lambda$ (where $\lambda$ is the minimum length of a transistor) per side. This result can be used with [1].
- The size of a four bit processor with a 10 bit sequence memory, including RAM look-up tables was found to require an area of about 400 by 900 $\lambda$. The grid size in the grid model would be larger than this, as SE wiring is not included in the estimate.

# Chapter 5
# Conclusions

## 5.1. Conclusions

The objective of this thesis is to show that although the complexity of the Viterbi Algorithm increases exponentially as the constraint length of the problem increases, a practical method for solving the problem exists. Through the use of a highly parallel structure, the Shuffle Exchange graph, the processing time is kept to one clock cycle per symbol interval, so that solutions to reasonably complex problems can be calculated in real time.

Of the six potential VA architectures considered for this thesis, the ones with most merit are the uniprocessor, the cascade, and the Shuffle Exchange. The uniprocessor solution has the slowest performance, but hardware can be designed to achieve the highest constraint length of the three layouts. The SE has the fastest performance, but is limited to small constraint lengths. The cascade occupies the middle ground with medium throughput and medium area. The SE layout was chosen for study because this balance was considered important.

The prototype designs show that the resulting layout is practical with present technology, and meets the objectives of high speed decoding. A prototype was constructed that is capable of solving constraint length two problems with a symbol interval of 500 nanoseconds. The designs also allowed the constants referred to in [1] to be approximated. A SE laid out on a grid model requires about 480 $\lambda$ square if metric RAM or ROM is not included. Approximately twice the area is required if the RAM is included.

The VLSI grid model does not deal with the problem that occurs when designs which are too big for a silicon die must be implemented. A strategy for expanding the problem size is described which is flexible in terms of design expansion and changes. Functional, Necklace and Exchange Edge partitioning are discussed, and edge partitioning is fully developed. Edge partitioning is a practical interim method

of expanding the size of SE graph that can be built, but the number of processors that will fit onto an edge chip will eventually outgrow the number of pins available to service them. Edge partitioning reaches a limit when no more pins can be added to a chip carrier to increase the number of processors per package and the circuit board layout size becomes impractically large due to a large package count.

Partitioning will be advantageous in expanding design sizes until the VLSI technology advances sufficiently to implement an entire SE layout of equivalent complexity on a single chip. However, whether partitioning or a single chip is used, a projection based on the minimum layout area of a SE graph, and the VLSI technology under development suggests that the SE will likely be restricted in the near future to solving problems with constraint lengths less than nine.

## 5.2. Recommendations

The following is a list of recommendations for further study.

- Examine the possibility of operating the 2-D array at or near the lower processing interval bound. If this is practical, this array processing scheme should be researched, as it has good decoding speed potential.
- The cascade layout shows potential for medium speed decoding. It would be useful to study its implementation in detail, to discover its performance limitations, and to see what the maximum practical memory length is.
- A decoder with memory length 5 should be implemented using the resubmitted edge chips. This would allow practical tests to be run on a SE of reasonable size.

# Appendix A
# Prototype CZZMB012

## A.1. Hardware Description

The pin functions are the following:

1 0b0: processor 0 transition metric B, bit 0

2 0b1: proc 0 metric B, bit 1

3 0b2: proc 0 metric B, bit 2

4 0b3: proc 0 metric B, bit 3

5 --- NC

6 CLK: clock input

7 1a3: proc 1 metric A, bit 3

8 1a2: proc 1 metric A, bit 2

9 1a1: proc 1 metric A, bit 1

10 1a0: proc 1 metric A, bit 0

11 1b0: proc 1 metric B, bit 0

12 1b1: proc 1 metric B, bit 1

13 1b2: proc 1 metric B, bit 2

14 1b3: proc 1 metric B, bit 3

15 --- NC

16 3a3: proc 3 metric A, bit 3

17 3a2: proc 3 metric A, bit 2

18 3a1: proc 3 metric A, bit 1

19 3a0: proc 3 metric A, bit 0

20 3b0: proc 3 metric B, bit 0

21 3b1: proc 3 metric B, bit 1

22 3b2: proc 3 metric B, bit 2

23 3b3: proc 3 metric B, bit 3

24 GND: power supply ground

25 --- NC

26 OUT: serial output

27 2a3: proc 2 metric A, bit 3

28 2a2: proc 2 metric A, bit 2

29 2a1: proc 2 metric A, bit 1

30 2a0: proc 2 metric A, bit 0

31 2b0: proc 2 metric B, bit 0

32 2b1: proc 2 metric B, bit 1

33 2b2: proc 2 metric B, bit 2

34 2b3: proc 2 metric B, bit 3

35 --- NC

36 VDD: power supply

37 0a0: proc 0 metric A, bit 0

38 0a1: proc 0 metric A, bit 1

39 0a2: proc 0 metric A, bit 2

40 0a3: proc 0 metric A, bit 3

It is important to note that due to the physical layout of the signal interconnections in the center of the chip, processors 0 & 1 have their A & B inputs reversed. This must be reflected in reversing the order of the transition metrics in the ROMs of the test jig.

When two numbers are compared, the least significant bit of B is ignored. If the rest of the number is the same as A, the select line is equal to the least significant bit of A. This approach is reasonable because if a tie exists, the choice is arbitrary.

## A.2. Channel Response

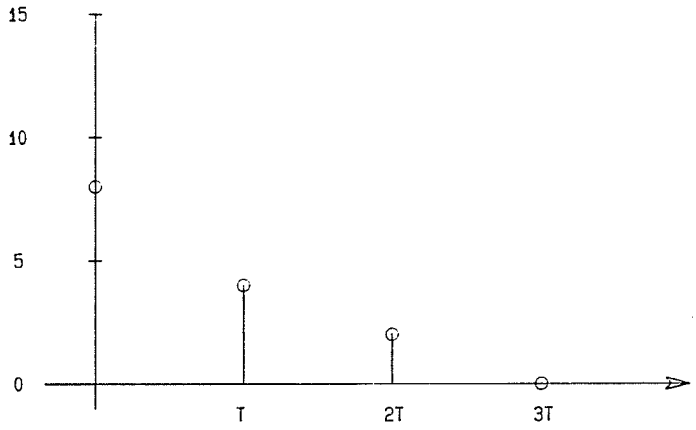


**Figure 30:** *Channel Response Graph*

## A.3. ROM Contents

Channel Metrics

| ROM | Addr | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0ab | 02 | 11 | 20 | 31 | 42 | 53 | 64 | 75 |
| 1ab | 8A | 79 | 68 | 57 | 46 | 35 | 24 | 13 |
| 2ba | 74 | 63 | 52 | 41 | 30 | 21 | 12 | 03 |
| 3ba | FD | EC | DB | CA | B9 | A8 | 97 | 86 |
| | 8 | 9 | A | B | C | D | E | F |
| 0ab | 86 | 97 | A8 | B9 | CA | DB | EC | FD |
| 1ab | 02 | 11 | 20 | 31 | 42 | 53 | 64 | 75 |
| 2ba | 14 | 25 | 36 | 47 | 58 | 69 | 7A | 8B |
| 3ba | 75 | 64 | 53 | 42 | 31 | 20 | 11 | 02 |

**Table 2:** *Test Jig Rom Contents - Channel Metrics.*

Test Patterns

| Addr | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|
| 0ab | 01 | 22 | 20 | 22 | 88 | 22 | 22 | 22 |
| 1ab | 11 | 22 | 22 | 12 | 88 | 22 | 20 | 22 |
| 2ba | 11 | 21 | 22 | 22 | 88 | 02 | 22 | 22 |
| 3ba | 11 | 22 | 22 | 22 | 48 | 22 | 22 | 21 |

**Table 3:** *Test Jig Rom Contents - Test Patterns.*

Note: The nibbles a,b, correspond to the trellis rather than the labels on the chip. An "a" transition is initiated by a 0, and a "b" by a 1. The port names were kept consistent from each processor, to the chip port names, and were wired to the ROMs with nibble "a" to bits 0-3, "b" to 4-7. The inputs are only reshuffled in the ROM content itself, as shown.

## A.4. Chip Test Sequence

The data sequence used to test the prototype is shown above the output which was produced.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 7 | 0 | 6 | 8 | 10 | 6 | 7 | 7 |
| 12 | 8 | 15 | 15 | 6 | 1 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | | | |
| X | X | X | X | X | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | |

The test patterns were produced with the following sequences of applied ROM addresses:

| Input Sequence | Output Pattern |
|---|---|
| 10 | 000000... |
| 11,12,13,11,12,13,... | 001001... |
| 14 | 111111... |
| 15,16,17,15,16,17,... | 011011... |

**Table 4:** *Test Pattern Generation.*

## A.5. Schematic Diagram

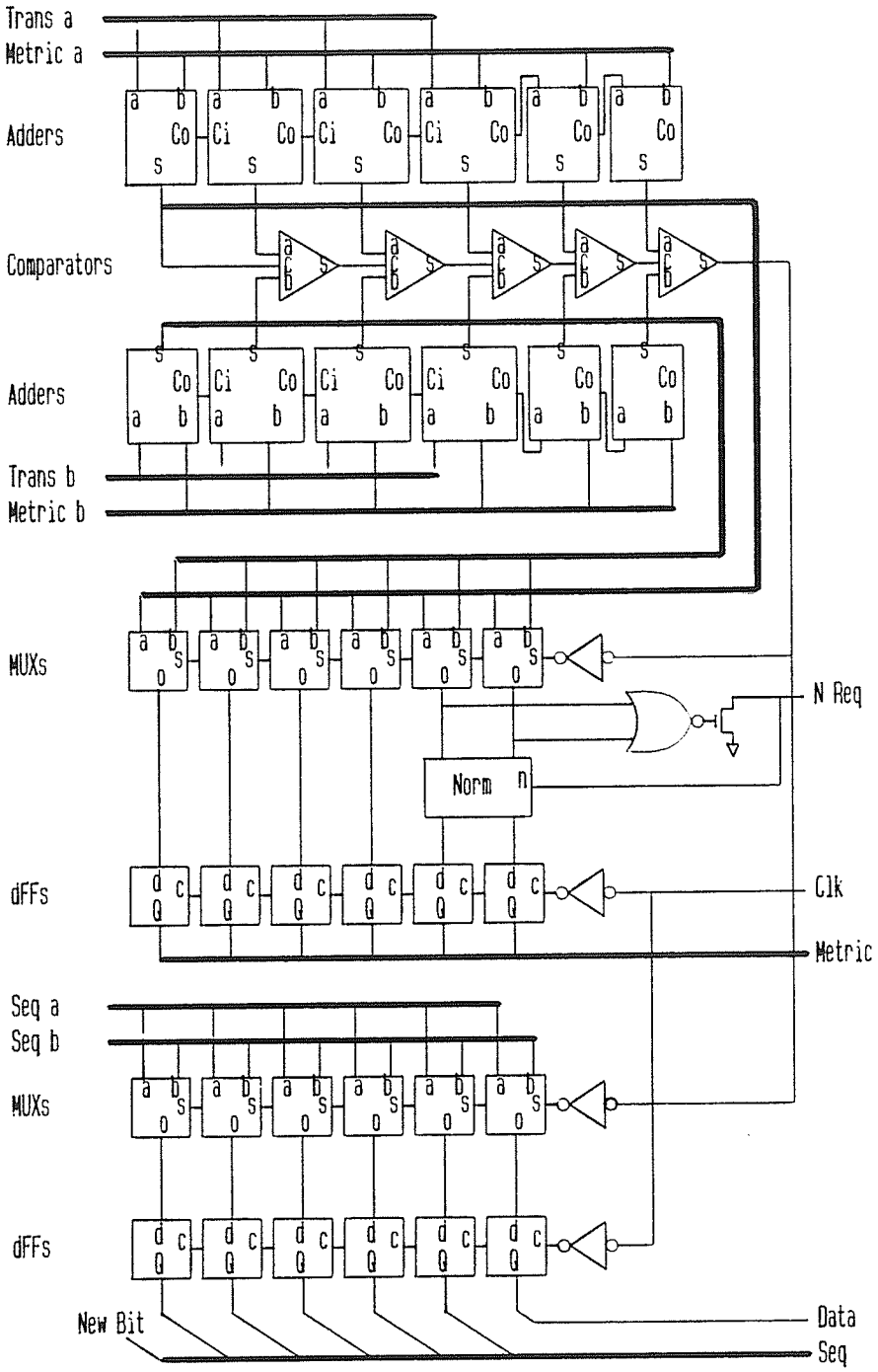


**Figure 31:** *Schematic of Prototype CZZMB012 Processor.*

# Appendix B
# Prototype IC3MB023

## B.1. Hardware Description

The pin functions are the following:

1 load: RAM load control. When reset is active, a 0 on this lead will zero the address counter. A 1 activates the counter, while enabling the t-gate and write line decoding to the 4 RAM sections. The counter presents all 16 addresses to each section in turn. The load line should be brought low once the last RAM location has been loaded.

2 r0: Received data input bit 0. 4 bit quantized data is applied to all chips in parallel. Data must be applied after positive clock transition, and must settle before negative edge. On negative clock transition, the value is latched and used as a look-up address for each transition metric. When reset and load are active, these inputs provide the RAM data for initialization. Data which has settled during the positive clock cycle, is written to a location and RAM section designated by the address counter during the negative clock cycle.

3 r1: Received data input bit 1.

4 r2: Received data input bit 2.

5 r3: Received data input bit 3.

6 mi00: State Metric Input Stream 0, Bit 0(2). A transmission from the even numbered processor preceeding this chip is labeled 0. The 4 bits in each number are multiplexed, low bits first during positive clock phase, high bit during negative clock. The ACS decision is latched on the next positive clock edge.

7 mi01: Metric in stream 0, bit 1(3).

8 mi10: Metric in stream 1, bit 0(2). A transmission from the odd numbered processor preceeding this chip is labeled 1.

9 mi11: Metric in stream 1, bit 1(3).

10 hi00: State History Input Stream 0, Bit 0(5). A transmission from the even numbered processor preceeding this chip is labeled 0. The 10 bits in each number are multiplexed, recent bits first during negative clock phase, previous bits during positive clock. The sequence chosen is latched on the next negative clock edge, 1/2 a clock cycle later than the metric decision.

11 hi01: History in stream 0, bit 1(6).

12 hi02: History in stream 0, bit 2(7).

13 hi03: History in stream 0, bit 3(8).

14 hi04: History in stream 0, bit 4(9).

15 hi10: State History Input Stream 1, Bit 0(5). Output from the odd numbered processor to this chip is labeled 1.

16 hi11: History in stream 1, bit 1(6).

17 hi12: History in stream 1, bit 2(7).

18 hi13: History in stream 1, bit 3(8).

19 hi14: History in stream 1, bit 4(9).

20 gnd: Power supply ground

21 clk: Single phase clock input

22 ho14: State History Output Processor 1, Bit 4(9). The survivor sequence chosen by processor 1 is latched and presented here in multiplexed form, newest bits first during the negative clock cycle, older bits later during the positive cycle. Thus, multiplexing between chips is synchronized.

23 ho13: History out proc 1, bit 3(8).

24 ho12: History out proc 1, bit 2(7).

25 ho11: History out proc 1, bit 1(6).

26 ho10: History out proc 1, bit 0(5). Bit 0 is the newest bit, and is always = 1 for proc 1.

27 ho04: State History Output Processor 0, Bit 4(9). The survivor sequence chosen by processor 0 is latched and presented here in multiplexed form.

28 ho03: History out proc 0, bit 3(8).

29 ho02: History out proc 0, bit 2(7).

30 ho01: History out proc 0, bit 1(6).

31 ho00: History out proc 0, bit 0(5). Bit 0 is the newest bit, and is always = 0 for proc 0.

32 mo11: State Metric Output Processor 1, Bit 1(3). The new sum latched by the ACS processor 1 is normalized if "ni" is active, and presented in a multiplexed format, least significant bits first, during the positive clock cycle, most significant during the negative cycle.

33 mo10: Metric out proc 1, bit 0(2).

34 mo01: Metric out proc 0, bit 1(3). The latched and optionally normalized sum from processor 0 is presented.

35 mo00: Metric out proc 0, bit 0(2).

36 d: Serial Data Output. The oldest bit in the sequence chosen by processor 1 is presented starting on the negative clock transition, and is valid for one entire cycle.

37 no: Normalize Output. An active high signifies that the two on chip processors agree that normalization is required. The "no" outputs from all chips must be anded together to detect when all sums are ready for normalization. The output becomes valid on the ACS decision latch.

38 ni: Normalize Input. When active high, the constant 4 is subtracted from all metric sums. Subtraction occurs asynchronously, and should be requested as soon as possible after the ACS decision latch, (positive clock edge) and settle before the negative clock edge. (In the case of IC3MB023, the constant 4 does not affect the least significant bits, so the normalization need only complete before the high order bits are latched by the receiving chip.)

39 res: Reset. Active high, the metric sums are set to zero on the next decision latch. The reset also reconfigures the RAM control for loading. The ram addresses now come from the load counter rather than the r0 to r3 inputs. While the load line is not active, the counter is continually reset to 0. When the load line is active, the r0 to r3 inputs are connected to the data lines on the RAM which is now loading.

40 vdd: Power supply.

## B.2. RAM Loading Procedure

For the purpose of loading the RAM, all inputs should be changed immediately following the positive clock transition, and should be constant for the rest of the clock cycle.

As it is convenient to feed all chips using the single received data bus, loading of the RAMs will take place sequentially. First, the reset line is raised for 1 cycle to reset all address counters. Then, beginning with one chip, that load line is asserted. The RAM values for processor 0 (of that chip) are fed in sequentially, starting with stream A, address 00, 01, ..., 0F (hex). The counter automatically advances to loading stream B, then processor 1, stream A, then B. When all 64 locations have been loaded, deactivate that chip's load line, and assert that of the next chip in the loading sequence. When all chips are initialized, release the reset line. This will free the input data stream to be interpreted as look-up addresses, and allow the metric sums to start accumulating.

## B.3. Schematic Diagrams

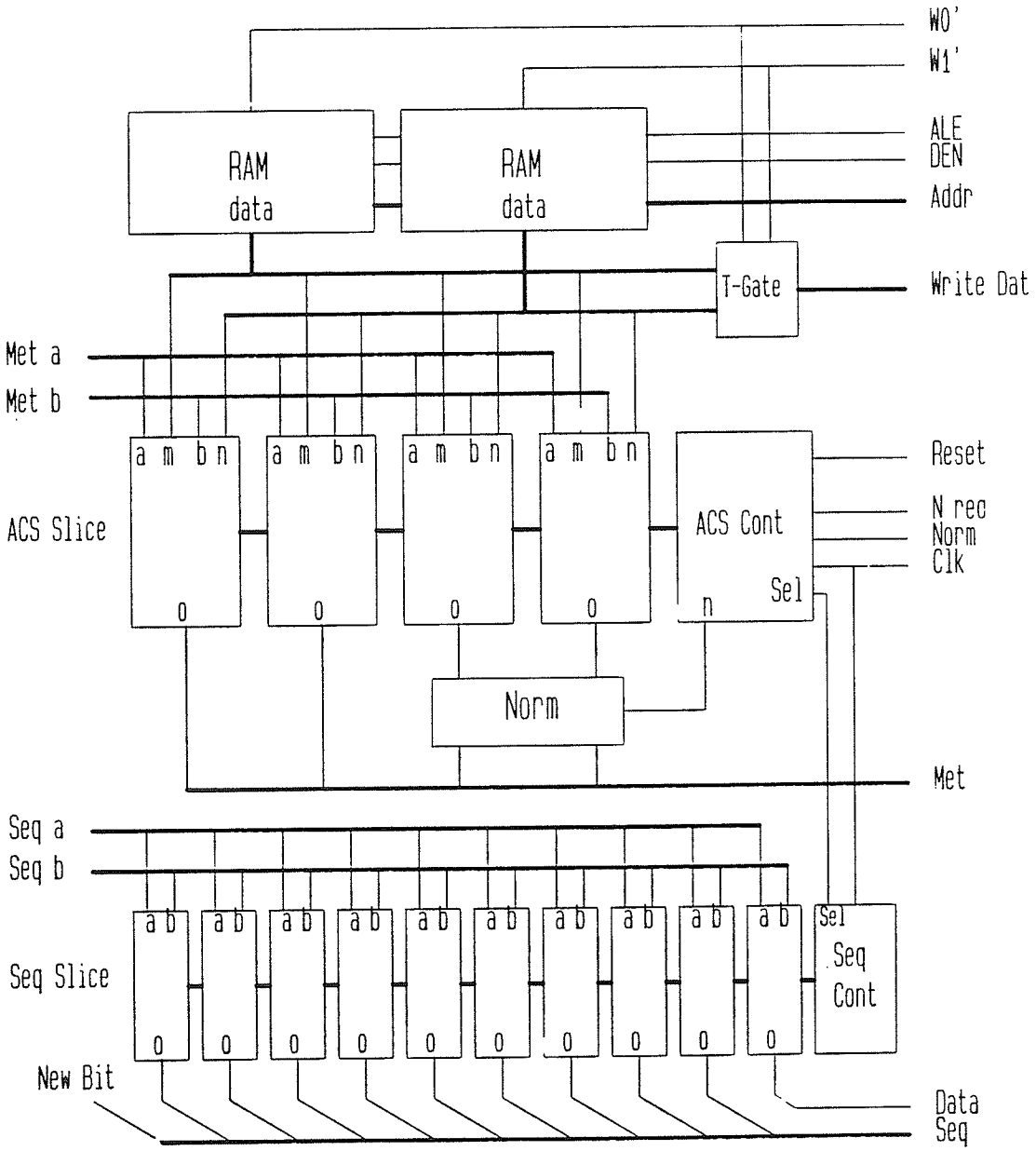


**Figure 32:** *Bit Slice Viterbi Processor.*

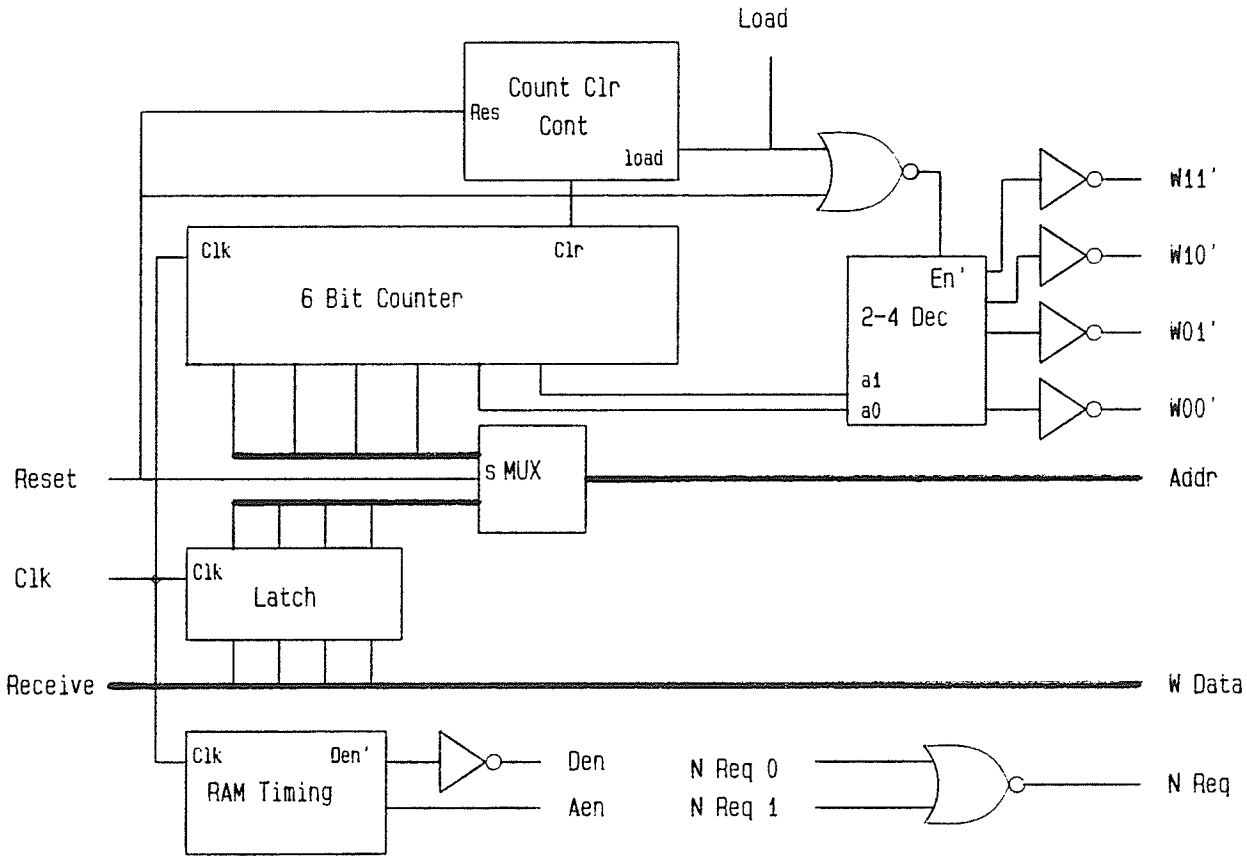


**Figure 33:** ***Ram Control.***

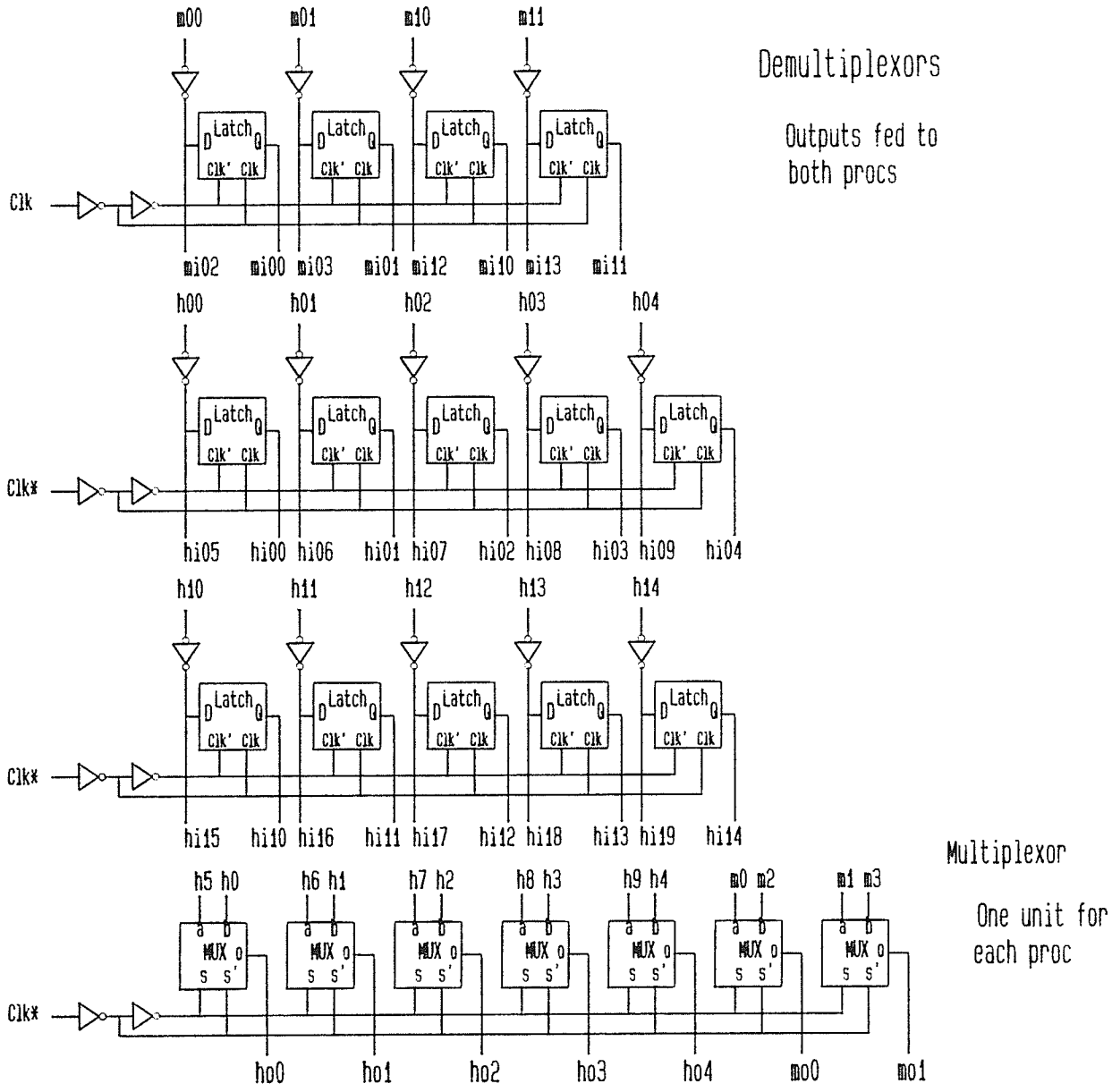


**Figure 34:** *Multiplexer, Demultiplexer.*

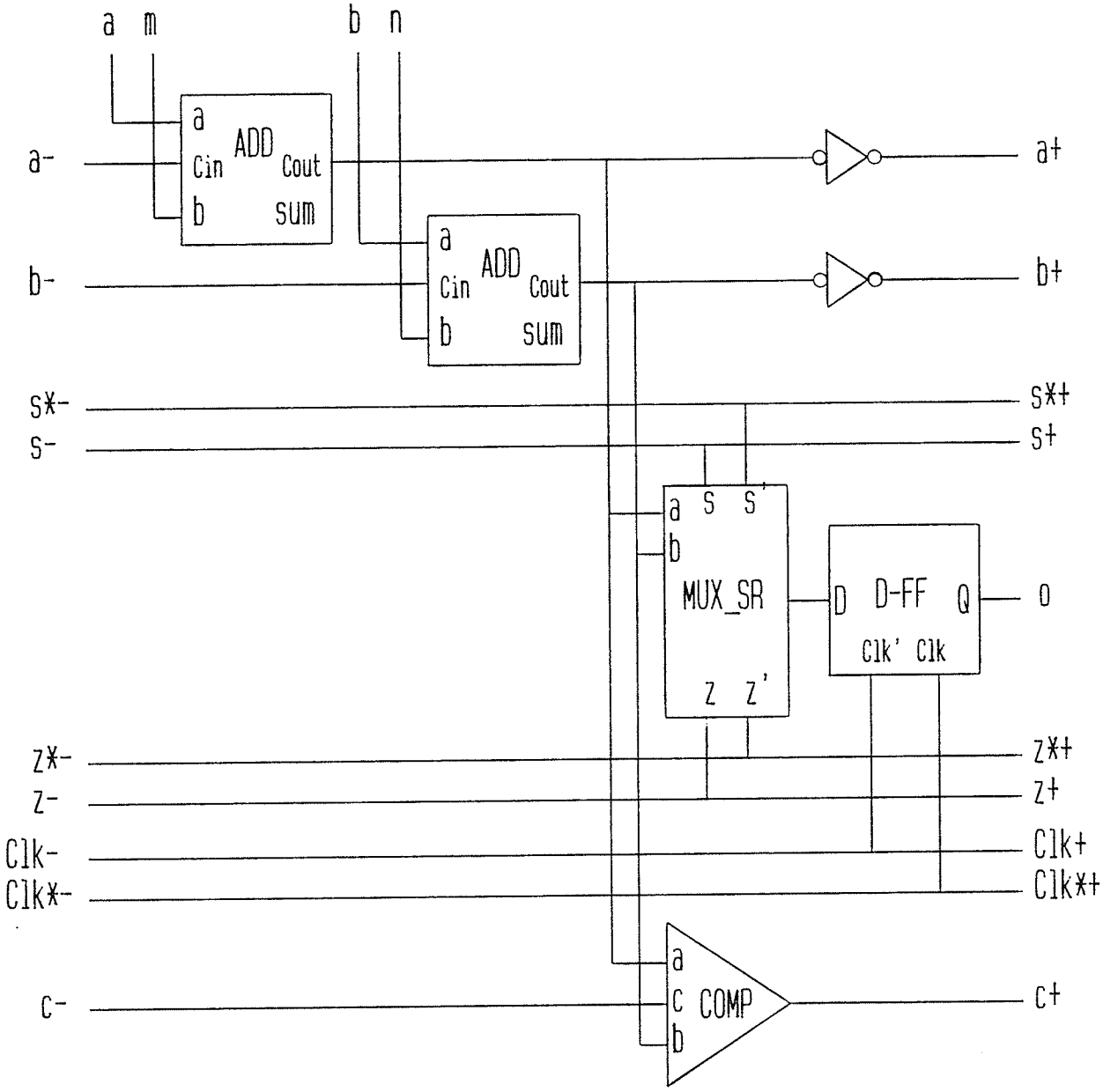


**Figure 35:** *ACS Slice.*

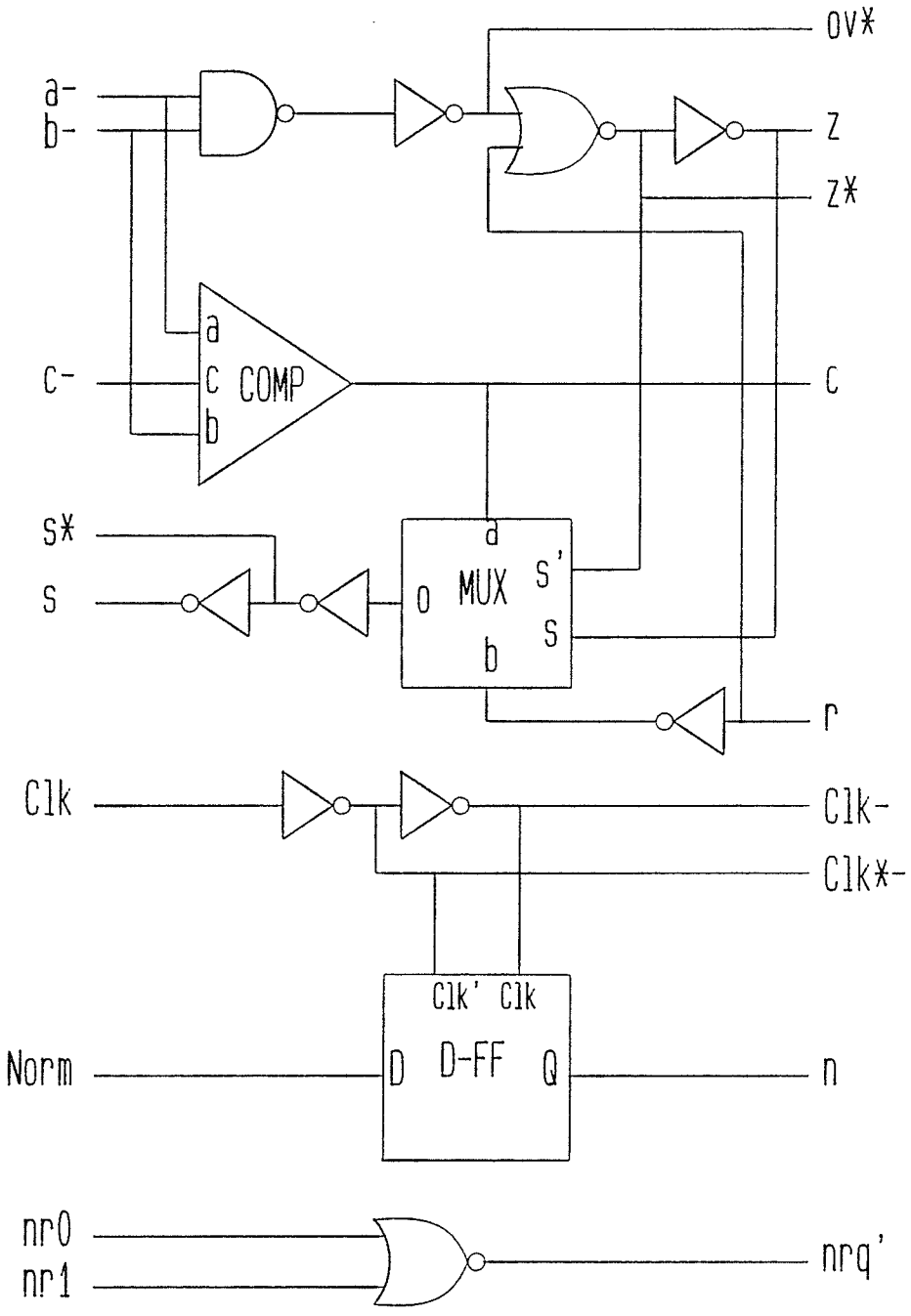


**Figure 36:** *ACS Control.*

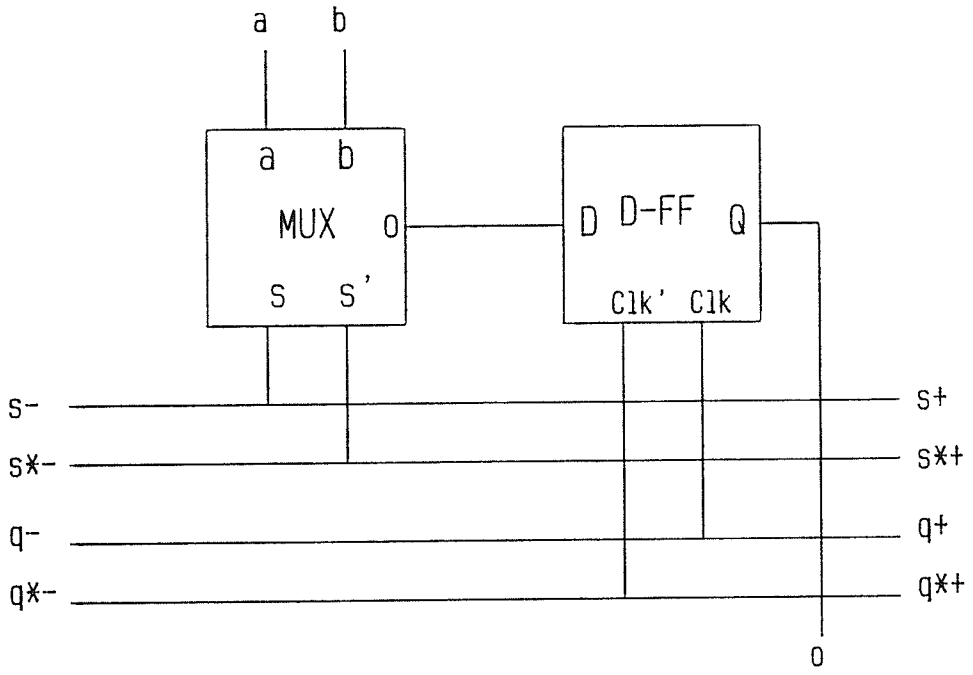


**Figure 37:** *Sequence Slice.*

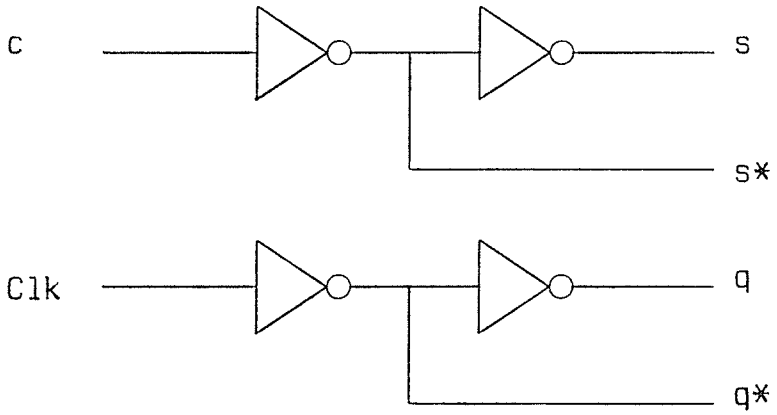


**Figure 38:** *Sequence Control.*

# References

[1] P. G. Gulak, "VLSI Structures for Digital Communication Receivers," *Ph. D. Dissertation, Dept. of Electrical Eng., University of Manitoba*, 1984.

[2] P. M. Chirlian, *Digital Circuits with Microprocessor Applications*, Matrix Publishers, Inc., Beaverton, Oregon, 1982, p. 190.

[3] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, Inc., New York, 1965, p. 528.

[4] A. J. Viterbi, J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, Inc., New York, 1979, p. 258,259.

[5] S. Lin, D. J. Costello, *Error Control Coding Fundamentals and Applications*, Prentice-Hall, Inc., 1983, ch 10, 11.

[6] P. G. Gulak, E. Shwedyk, "VLSI Structures for Viterbi Receivers: Part 1 - General Theory and Applications." *IEEE Journal on Selected Areas in Communications*, Vol. SAC-4, No. 1, Jan 1986, p. 142-154.

[7] H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers*, Vol. C-20, No. 2, Feb 1971, pp. 153-161.

[8] C. D. Thompson, "A Complexity Theory for VLSI", *Ph. D. thesis, Dept. of Computer Science, Carnegie Mellon University*,1980.

[9] F. T. Leighton, and G. L. Miller, "Optimal Layouts for Small Shuffle-Exchange Graphs," *VLSI 81*, J. P. Garry (Editor), Academic Press, p. 289-300.

[10] D. Kleitman, F. T. Leighton, M. Lepley, G. L. Miller, "New Layouts for the Shuffle-Exchange Graph," *Proceedings of the 13th ACM Symposium on Theory of Computing*, May 1981, p. 278-292.

[11] R. W. Keyes, "The Evolution of Digital Electronics towards VLSI." *IEEE Journal of Solid State Circuits*, Vol. SC-14, Apr. 1979, pp. 193-201.

[12] S. M. Rubin, *An Integrated Aid for Top-Down Electrical Design*," Fairchild Laboratory for Artificial Intelligence Research, Palo Alto California.

[13] R. D. Schneider, "An Interactive Digital MOS Timing Simulator with an APL User Interface," *M. Sc. Thesis, Electrical Eng. Dept., University of Manitoba*, 1985.